GPC-4301

GP-IB インタフェースモジュール用 Windows ドライバ

Help for Windows

www.interface.co.jp

目　次

第1章　はじめに

1.1　概要

GPC-4301は、Windows上のアプリケーションから、弊社GP-IBインタフェース製品を制御する為のソフトウェアです。
弊社GP-IBインタフェース製品をWindows上のアプリケーションからDLLをダイナミックリンクし、APIをコールすることにより制御します。
　本ドキュメントは、Windows上でGPC-4301を使用するための情報を掲載しています。

1.2　特長

1.2.1　高機能版DLL

●コントローラ・イン・チャージにおいて、データ送信/受信時におけるトーカ、リスナの指定をGP-IBバスコマンドにて事前に行う必要はありません。わずらわしいトーカ、リスナの指定はドライバ側にて自動で設定するため、機器のGP-IBアドレスの指定のみでデータの送信/受信を実行することができます（トーカ、リスナを意識しないデータ送受信）。
●1枚だけではなく、弊社GP-IBインタフェースを混在にて最大16枚まで制御することができます（複数枚サポート）。
●データ転送終了まで待たずに別の処理を行うことができます（非同期入出力サポート）。
●非コントローラとしての使用も可能なため、弊社GP-IBインタフェースを制御機器の代わりにして、コントローラから制御させる場合にも使用できます。
●SRQ受信時のイベント、およびコールバック関数の呼び出しをサポートしています。
●トーカ、リスナのみならず、デバイストリガ、デバイスクリア、IFC受信など、多種多様な状態を検出することができます（非コントローラ時）。
●ドライバの様々な動作設定（デリミタ、GP-IBアドレス等）をソフトウェアにより動的に変更/取得可能です。
●1次アドレスのみだけでなく、1次/2次アドレスも使用可能です。

1.2.2　標準版DLL

●GP-IB制御に必要最小限の関数を用意しています。
●SRQ受信時に、イベント、およびコールバック関数を呼び出すことができます。

第2章　製品仕様

本ソフトウェアは、弊社仕様APIのWindows用ドライバソフトウェアです。
IEEE 488.1/IEEE 488.2 API、およびNational Instruments社LabVIEW上から弊社GP-IBインタフェースモジュールを制御したい場合は、弊社Web siteより、GPC-4301Nをダウンロードし、インストールしてください。

また、GPC-4301Nは本ソフトウェア（GPC-4301）との併用はできません。
誤って本ソフトウェアをインストールしてしまった場合は、GPC-4301NのReadmeを参照し、ドライバの更新を行ってください。

2.1　基本仕様

項目	仕様
最大デバイス数	16枚
入出力チャンネル数	1チャンネル
入出力形式	IEEE-488.1規格(GP-IB)準拠 IEEE-488.2対応
マイアドレス設定	1次アドレスのみ 1次アドレス、2次アドレス 上記、2つのアドレスモードをソフトウェアにて設定可能
送信/受信デリミタ	・デリミタ無し ・EOIのみ ・CRのみ ・CR+EOI ・LFのみ ・LF+EOI ・CRLF ・CRLF+EOI ・NULL (00h)　（※1） ・"!x"感嘆符に続く文字をデリミタとする　（※1） 上記から任意選択可能
データ転送速度	最大1.1 MB/s （バスマスタ転送。使用環境および機器の速度に依存します。）

※1　高機能版DLLの機能です。

2.2 インタフェース・ファンクション

Function	機能	内容
C	C1	システム・コントローラ機能
	C2	IFC送信、コントローラ・イン・チャージ機能
	C3	REN送信
	C4	SRQ応答
	C5	インタフェース・メッセージ送信、コントロール受け、コントロール渡し、自分自身へのコントロール渡し、パラレル・ポール、ハンドシェークに同期してコントロール（※1）
	C28	インタフェース・メッセージ送信（※2）
SH	SH1	ソース・ハンドシェイク全機能
AH	AH1	アクセプタ・ハンドシェイク全機能
T	T6	基本的トーカ、シリアル・ポール、MLAによるトーカ解除
TE	TE6	基本的拡張トーカ、シリアル・ポール、MSAによるトーカ解除
L	L4	基本的リスナ、MTAによるリスナ解除
LE	LE4	基本的拡張リスナ、MSAによるリスナ解除
SR	SR0	サービス・リクエスト機能無し（※2）
	SR1	サービス・リクエスト全機能（※1）
RL	RL0	リモート・ローカル機能無し（※2）
	RL1	リモート・ローカル全機能（※1）
PP	PP0	パラレル・ポール機能無し（※2）
	PP1	リモート・コンフィギュレーションによるパラレル・ポール（※1）
	PP2	ローカル・コンフィギュレーションによるパラレル・ポール（※1）
DC	DC1	デバイス・クリア全機能
DT	DT1	デバイス・トリガ全機能

※1 高機能版DLLの機能です。
※2 標準版DLLの機能です。

2.3 製品構成

製品構成		ファイル名	説明
弊社管理用ファイル		GPC4301.VER	弊社ソフト管理用ファイル
最新情報ドキュメント		README.HTM	最新ドキュメント掲載ファイル
インストールプログラム		SETUP.EXE	インストール用ファイル
動作確認プログラム		GpibUtil.EXE	動作確認用プログラム
		GpibStart.EXE	GP-IBコマンドライン開始プログラム
		GpibStop.EXE	GP-IBコマンドライン停止プログラム
		GpibWrite.EXE	コマンドラインデータ送信プログラム
		GpibRead.EXE	コマンドラインデータ受信プログラム
		GpibClear.EXE	コマンドラインデバイスクリア実行プログラム
		GpibTrigger.EXE	コマンドラインデバイストリガ実行プログラム
		GpibSpoll.EXE	コマンドラインシリアルポール実行プログラム
自己診断プログラム		DiagGpib.EXE	自己診断プログラム
サンプルプログラム	Visual C# .NET	*.cs	Visual C# .NET用サンプルプログラム
	Visual C++	*.c	Visual C++(MFC)用サンプルプログラム
	Visual Basic .NET	*.vb	Visual Basic .NET用サンプルプログラム
	Visual Basic	*.bas	Visual Basic用サンプルプログラム
	Delphi	*.pas	Delphi用サンプルプログラム
コントロールパネル		GPC4304.CPL	コントロールパネルアプレット
		IFCardId.CPL	コントロールパネルアプレット（CardBusシリーズのみ）
DLL（高機能版）		GPC4304.DLL	ダイナミックリンクライブラリファイル
		GPC4304.LIB	インポートライブラリファイル
DLL（標準版）		GPC43042.DLL	ダイナミックリンクライブラリファイル
		GPC43042.LIB	インポートライブラリファイル
デバイスドライバ		GPC4301.SYS	Windows 2000/NT以降のOS 用ドライバ
		GPC4301.VXD	Windows Me/98/95用ドライバ
		GPC4301.INF CBI4302.INF CBI4321.INF	ドライバインストールファイル
		GPC4301.SLD CBI4302.SLD CBI4321.SLD	Windows Embedded用ドライバ　SLDファイル
		IFCardId.SYS	Windows 2000/NT以降のOS 用ドライバ（CardBusシリーズのみ）
		IFCardId.VXD	Windows Me/98/95用ドライバ（CardBusシリーズのみ）
ヘッダファイル		GPC4304.H	Visual C++用ヘッダファイル（高機能版）
		GPC43042.H	Visual C++用ヘッダファイル（標準版）
		GPC4304.BAS	Visual Basic用ヘッダファイル（高機能版）
		GPC43042.BAS	Visual Basic用ヘッダファイル（標準版）
		GPC4304.PAS	Delphi用ヘッダファイル（高機能版）
		GPC43042.PAS	Delphi用ヘッダファイル（標準版）
Help		HELP.PDF	Help
		HELP_NET.PDF	.NET用補足Help

※Visual C# .NET，Visual Basic.NET用サンプルプログラムは、それぞれViausl C# .NET 2003，Visual Basic .NET 2003を使用して作成しています。

第3章　導入方法

3.1　インストール手順

README.HTMのインストール方法を参照してください。

3.2　コントロールパネル

コントロールパネルの設定により、デバイス使用時の初期状態を指定できます。
インストール直後の動作モードおよびドライバの状態は下図の通りです。

パラメタ	説明	インストール直後の初期値
ボード番号	インタフェースモジュール上のRSW1設定値。	0
動作モード	コントローラ動作をさせるためには「MASTER」、非コントローラ動作をさせるためには「SLAVE」を設定します。	MASTER
1次アドレス	00h～1Ehの範囲内で、GP-IBバスに接続している他の機器と重ならないアドレスを設定します。	00（h）
2次アドレス	60h～7Ehの範囲内で、アドレスを設定します。 FFhを設定すると、2次アドレスは無しとなります。	FF（h）
送受信 タイムアウト	1～65535の範囲内で、機器とのデータ転送時間を考えて、十分余裕のある時間を設定します。	100（×100ms）
事象変化 タイムアウト	1～65535の範囲内で、非コントローラ側の各事象変化（トーカ指定/リスナ指定/デバイストリガ受信などが検出されるまで待つタイムアウト時間を設定します。	100（×100ms）
コマンド送出 タイムアウト	1～65535の範囲内で、マルチラインメッセージ，バスコマンドの送出タイムアウト時間を設定します。	100（×100ms）
ハンドシェイク タイミング	応答を取りあうまでの時間を「超高速（350ns）」，「高速（500ns）」，「通常（2μs）」から選択します。（マルチラインメッセージセットリングタイム T1）	超高速
送信デリミタ	GP-IB制御を行う機器のデリミタと一致させるようにします。ご使用の機器の取扱説明書を参照してください。	CRLF＋EOI
受信デリミタ		CRLF＋EOI
STB応答時間	1～65535の範囲内で、測定器からステータスバイトが送られてくるのを待つ時間を設定します。	10（×100ms）
PP実行時間	パラレルポールを行うときの待ち時間を「2μs」，「10μs」から選択します（パラレルポール実行時間T6）。光ケーブルを使用してパラレルポールを実行する場合は、「10μs」に変更します。	2μs

ドライバパラメタの初期値を必要に応じて変更してください。
ドライバパラメタ（ボード番号を除く）は、プログラム実行時に変更可能です。

ドライバパラメタの初期設定方法を以下に示します。

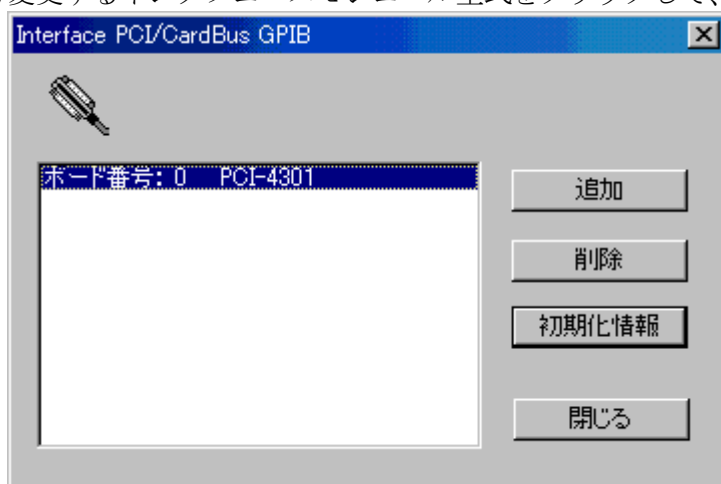
1) 「コントロールパネル」を起動し、「Interface GP-IB Configuration」をダブルクリックします。

※　ソルコン Classembly Devices®、I/O付きタッチパネル Classembly Devices®のGP-IBモデルをお使いの場合、型式は「PCI-432601」と表示されます。

【ボタンの確認】

追加　：　インタフェースモジュールの追加を行います(通常は使用する必要はありません)。
削除　：　インタフェースモジュールの削除を行います。
初期化情報　：　現在登録されているインタフェースモジュールの初期化情報を設定します。
閉じる　：　終了します。

※削除、初期化情報のボタンは、処理を行うインタフェースモジュールを選択してからクリックしてください。

2) 変更するインタフェースモジュール型式をクリックして、選択状態にしてください。

3) 初期化情報の設定(変更)

選択したインタフェースモジュールのドライバパラメタ設定（変更）を行います。

「Interface GP-IB Configuration」ダイアログボックスの「初期化情報」ボタンをクリックすると、初期情報設定ダイアログボックスが表示されます。

※ボード番号は、インタフェースモジュール上のロータリスイッチRSW1(CardBus製品はCardBus ID)と同じ値に設定してください。
ソルコン Classembly Devices®、I/O付きタッチパネル Classembly Devices®のGP-IBモデルをお使いの場合は、0を指定してください。

※1次アドレスは00h～1Ehの範囲内で、そのGP-IBバスに接続されている他の機器と、重複しない値に設定してください。

※2次アドレスは60h～7Ehの範囲内で、そのGP-IBバスに接続されている他の機器と、重複しない値に設定してください。また、FFhを設定しますと、GP-IBインタフェースモジュール自身の2次アドレスは無しとなります。

※送信/受信デリミタは、GP-IB制御を行う機器のデリミタと一致させるようにしてください。

※送受信タイムアウト時間は、機器とのデータ転送時間を考えて、十分余裕のある時間を設定してください。

※コマンド送出タイムアウト時間は、マルチラインメッセージ送信時に機器がタイムアウトする場合、値を増やすようにしてください。

※光ケーブルを使用してパラレルポールを実行する場合は、PP実行時間を10μsに変更してください。

※ハンドシェイクタイミングは、機器とのハンドシェイクを行う際の信号待ち時間を設定します。

- 「超高速」 ： 350ns (デフォルト)
- 「高速」 ： 500ns
- 「通常」 ： 2μs

デフォルト状態にて送信/受信時に「送信エラー：-13」または「タイムアウト：-14」が発生する場合には、ハンドシェイクタイミングの速度を変更してみることをお勧めします。

4) 設定変更後、再起動を行います。
再起動後、変更した情報がドライバに反映されます。

■非コントローラで使用する場合
「動作モード」を「SLAVE」に変更してください。

■ボードアクセス番号とインタフェースモジュールの関係
ボードアクセス番号は、インタフェースモジュールを識別するための番号で、インタフェースモジュール上にあるボードID設定用ロータリスイッチ(RSW1)の番号(CardBus製品はCardBus ID)を指定します。

このボードアクセス番号は、「初期化情報」で設定し、各インタフェースモジュールと対応付けされます。1台のコンピュータで複数枚のインタフェースモジュールを使用する場合は、このボードアクセス番号を重複しないように設定してください。

同時に使用できるインタフェースモジュール枚数は、PCIシリーズ/CompactPCIシリーズ/CardBusシリーズの各GP-IBインタフェースモジュールを合せて最大16枚までです。

3.3 動作確認(動かしてみよう)

この節では、本ソフトウェアに含まれているGP-IBユーティリティを用いて、例としてPCI-4301とGP-IBで接続された計測機器との間の基本的なバス動作を確認してみましょう。

※以下の準備をしてください。

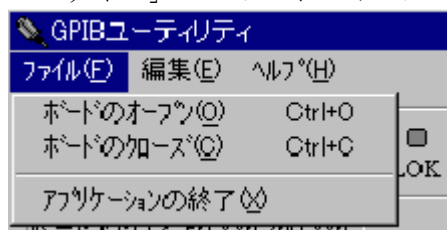
1.PCI-4301を、GP-IBケーブルでGP-IB機器と接続します。
　ユーザーズマニュアル『機器への接続方法』を参照してください。
2.PCI-4301のドライバのインストール、コントロールパネルの設定を行います。
　『3.2　コントロールパネル』を参照してください。動作モードは「MASTER」に設定します。
3.PCI-4301のソフトウェアのインストールを行います。
　README.HTMのインストール方法を参照してください。
4.接続された計測機器の、機器アドレスをご確認ください。

確認方法は、計測機器のマニュアルをご覧ください。

計測機器の機器アドレス※1は、インタフェースモジュールの機器アドレス(プライマリアドレス※2)と重複しないようにしてください。重複していた場合には、計測機器側もしくはインタフェースモジュール側のどちらかの機器アドレスを変更してください。

3.3.1 ユーティリティ起動とインタフェースモジュールの初期化

GP-IBユーティリティの起動と、PCI-4301を初期化します。

1) 「スタート」メニューより「プログラム」－「Interface GPC-4301」－「GpibUtil」を起動します。
2) 「ファイル」－「ボードのオープン」を選択します。

3) 「ボードのオープン」ダイアログボックスが表示されます。
使用するインタフェースモジュール，インタフェースモジュールの機器アドレス，計測機器のデリミタを設定し、OKボタンをクリックします。
●「ボード」リストには、アクセスできるGP-IBインタフェースモジュールの型式とインタフェースモジュール識別番号の一覧が表示
　されます。このリストからアクセスしたいインタフェースモジュールを一つ選択します。
　(ユーザーズマニュアル『インタフェースモジュール識別ロータリスイッチ(RSW1)』参照)
●「プライマリアドレス」，「セカンダリアドレス※2」は、選択したインタフェースモジュールの機器アドレスを設定します。初期値はコントロールパネルで設定した値になっています。
　※16進数で入力してください。
●「デリミタ」は、接続する計測機器のマニュアルを参照し、一致するものを設定します。

以上で、PCI-4301がGP-IBユーティリティで使用できるようになりました。

次に、計測機器をGP-IBの手順によって初期化します。

4) 「IFC」ボタンをクリックして、IFC※3メッセージを発行します。
IFCメッセージ発行によって、GP-IBに接続されているすべての機器のGP-IBインタフェースは初期化されます。
IFCを発行すると、本インタフェースモジュールが当番コントローラ※4になり、CICランプが点灯します。
同時に、ATNランプも点灯します。

5) REN※5メッセージを発行します。
「REN」ボタンをクリックして、RENランプを点灯させせます。
この時、本インタフェースモジュールはRENメッセージを有効にしています。

6) SDC※6メッセージを発行します。
1.「Device Clear」ボタンをクリックします。
2.「Address」ボックスに計測機器のアドレスを16進数で入力します。
3.最後に「Clear」ボタンを押します。
SDCメッセージによって、GP-IBインタフェースに接続されている計測機器自体の初期化が行われます。
以上で、GP-IB機器の初期化が完了しました。

【用語解説】

※1 機器アドレス

計測機器，GP-IBインタフェースモジュールに割り当てるGP-IBアドレスのことです。
GP-IBの制御を行うとき、この機器アドレスを指定することで通信相手を特定します。
インタフェースモジュールと計測機器の機器アドレスは重複しないように設定してください。
GP-IBユーティリティ側では、「ボード機器アドレス」として、インタフェースモジュール自身のアドレスを表示しています。

※2 プライマリアドレス、セカンダリアドレス

機器アドレスの種類として、次の2種類があります。

・プライマリアドレス(1次アドレス) ：16進数00h～1Eh（10進数では0～30）の値が設定可能です。
・セカンダリアドレス(2次アドレス) ：16進数60h～7Eh（10進数では96～126）の値が設定可能です。

大半の計測機器はプライマリアドレスのみ指定します。
これは、1つのアドレス指定で計測機器を特定可能だからです。
しかし、計測機器によっては、セカンダリアドレスを持つものもあります。
これは、1台の計測機器の中に複数の機能を持っている場合などに使用します。
(例：マルチメータとロジックアナライザーが1台の計測機器となっており、機能を切替えて使用する場合など)

セカンダリアドレスのみ単独で指定することはできません。
必ず、プライマリアドレスとセカンダリアドレスはペアで指定します。

・プライマリアドレス0Ahを指定 →1つの計測機器を特定
・セカンダリアドレス60hを指定 →プライマリアドレスで特定した計測機器の中で更に機能を特定

※3 IFC（Interface Clear）

システムコントローラが、すべての機器のインタフェース機能（コントローラ，トーカ，リスナ機能）を初期化するために送信するメッセージ。

※4 コントローラ

どの機器がトーカやリスナになるかを指定したり、バスの制御機能を持つもの。コンピュータの中の管理者，会議の進行役にあたる。

※5 REN（Remote Enable）

システムコントローラが、相手機器をGP-IBバスを経由して制御できるよう（リモートモード）に指示する時に送信するメッセージ。

※6 SDC（Selected Device Clear）

リスナに指定された機器を、初期状態に戻すコマンドです。機器により、初期状態の意味は異なります。

3.3.2 データの送信

計測機器にデータを送信してみましょう。

計測機器の中には、計測を行う時にデータ（デバイスディペンデントメッセージ）を要求するものがあります。
データの意味やその後の動作は、個々の計測機器によって定義されています。
たとえば、マルチメータの場合は、計測トリガや入力レンジなどを、データを与えることによって設定できます。

1) 1.「Data Send」ボタンをクリックします。
   2.「Address」ボックスに相手機器のアドレスを16進数で入力します。
   3.「Exec」ボタンをクリックします。
   PCI-4301がトーカ※1になったことは、TAランプが点灯したことで確認でき、データが送信できる状態になります。

2) 「Send Data」テキストボックス内に、リスナ※2へ送信するデータをASCII文字で入力し、「Enter」キーを押すと、リスナに対してデータを送信します。

【用語解説】
※1 トーカ
送信機能を持つもの（話し手）。
GP-IBでは接続している機器が、勝手にデータを送信することができない。コントローラによりトーカに指名されると、データを送信することができる。

※2 リスナ
受信機能を持つもの（聞き手）。
GP-IBでは接続している機器が、勝手にデータを受信することができない。コントローラによりリスナに指名されると、データを受信することができる。

3.3.3 データの受信

計測機器が出力するデータを受信してみましょう。

1) 1.「Data Receive」ボタンをクリックします。
2.「Address」ボックスに相手機器のアドレスを16進数で入力します。
3.「Exec」ボタンをクリックします。
PCI-4301がリスナになったことは、LAランプが点灯したことで確認でき、データが受信できる状態になります。

2) データを受信します。
「Recv」ボタンをクリックします。クリックすると、計測機器が出力するデータをデリミタま

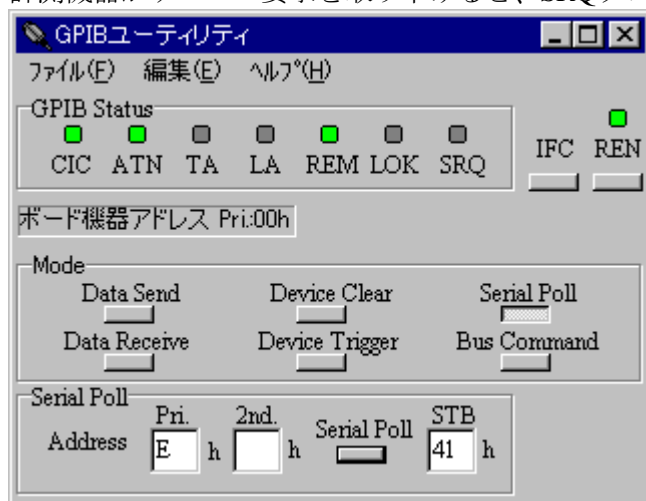
で受信し、「Receive Data」テキストボックス内に受信データを表示します。

3.3.4 シリアルポールの実行

計測機器の中には、サービス要求のためにSRQ※1メッセージを有効にし、ステータスバイト（STB）※2の受け取りを要求するものがあります。コントローラがSTBを受信することをシリアルポール※3といいます。

1) 計測機器がサービス要求するように制御します。
計測機器によって、サービス要求する条件が違います。計測機器のマニュアルを参照してください。

計測機器がサービス要求を行うと、SRQメッセージが有効になります。SRQランプが点灯することにより確認できます。

2) シリアルポールを実行します。
   1.「Serial Poll」ボタンを押します。
   2.「Address」ボックスに相手機器のアドレスを16進数で入力します。
   3.「Serial Poll」ボタンをクリックします。
   クリック後、計測機器から受信されたSTB値が「STB」ボックスに表示されます。
   計測機器がサービス要求を取り下げると、SRQランプは消灯します。

※ユーティリティの詳細（その他の機能等）については『ユーティリティプログラム』を参照してください。
以上で、PCI-4301を使ったGP-IBの基本的バス動作を確認できました。

【用語解説】

※1 SRQ
サービスリクエスト。
機器からの事象の変化を、コントローラに伝えるための仕組みです。事象の変化を受け取るために「割り込み」を用いますが、サービスリクエストも同じことを実現します。

※2 ステータスバイト(STB)
機器の状態を示す1バイトのデータ。
コントローラからのシリアルポールによって読み取られます。通常、SRQを発行した場合、そのSRQの発行理由を、このSTBによって知ることができます。

※3 シリアルポール
コントローラが機器に対して、サービスリクエストを送信したかどうか、1台ずつ順番に問い合わせ、ステータスバイトを取得する処理です。

3.3.5 終了方法

インタフェースモジュールの使用を終了する場合は、「ファイル」－「ボードのクローズ」を選択します。
本ユーティリティを終了する場合は、インタフェースモジュールがクローズされている状態で、「ファイル」－「アプリケーションの終了」を選択します。

3.4 高機能版DLL実行手順

3.4.1 コントローラとしての実行手順

GP-IB製品を複数枚使用する場合は、
・Compact PCI, PCI製品の場合、インタフェースモジュール上のロータリースイッチRSW1の設定値
・CardBus製品の場合、CardBus IDの設定値
がGP-IB製品同士で重複しないように設定してからスロットに実装し、システムを起動して下さい。GP-IB製品が複数存在する場合、制御対象を一意に識別するための番号となります。重複していた場合、本ソフトウェアは正常に動作いたしません。

インタフェースモジュールをコントローラとして使用する場合の基本的な制御の手順は以下の通りです。（記述例はC言語です）

1. 初期化
インタフェースモジュールをPciGpibExInitBoard関数で初期化します。このとき、アプリケーションのウィンドウハンドルも一緒に指定します。非同期で送受信を行わない場合や、メッセージポストを行わない場合はWM_NULLまたは0を指定します。

```
PciGpibExInitBoard(0, hWnd);
```

初期化が正常終了後、指定したボードアクセス番号(この場合は 0)で以後の各関数を使用することができます。

2. インタフェースモジュールの設定
PciGpibExSetConfig関数で、インタフェースモジュールのデリミタ等の各種設定を行います。この関数を実行しない場合は、初期値で設定されます。初期値については、コントロールパネルアプレットで設定した値が使用されます。
また、この関数は初期化(PciGpibExInitBoard関数)の実行後は何度でも実行できます。
※コントロールパネルアプレットの説明については、READMEおよびユーザーズマニュアルを御参照ください。

```
PciGpibExSetConfig(0, "/SDELIM=EOI /RDELIM=CRLF+EOI");
```

送信デリミタとして[EOIのみ]、受信デリミタとして[CRLF+EOI]を設定しました。

3. GP-IBインタフェースの初期化
コントローラの場合は、PciGpibExSetIfc関数を使用してGP-IBバスの初期化を行います。
続いて、PciGpibExSetRen関数を使用してREN信号を有効にします。

```
PciGpibExSetIfc(0, 1);
PciGpibExSetRen(0);
```

4. 送信

機器固有の制御コマンドは GP-IB バスのデータ送信として実行します。
PciGpibExMastSendData 関数で指定した機器に対してデータを送信します。
・1 番目の引数はボードアクセス番号です。
・2 番目の引数はデータ送信先の機器アドレステーブルを指定します。
・3 番目の引数は送信データ長(単位:バイト)です。
・4 番目の引数は送信データへのポインタです。
・5 番目の引数は同期転送時には WM_NULL または 0 を指定します。非同期転送の場合にはデータ送信完了時にこの引数に設定したメッセージコードがポストされます。

```
int nAdrsTbl[8];
       ・
       ・
nAdrsTbl[0] = 1;     // 制御する測定機器のアドレス
nAdrsTbl[1] = -1;    // アドレステーブルの終端
                     // 機器アドレステーブルの終端には必ず -1 を代入してください。
       ・
       ・
PciGpibExMastSendData(0, nAdrsTbl, 10, ″0123456789″, 0);
```

5. 受信

機器から送られてくるデータは PciGpibExMastRecvData 関数にて受信します。
PciGpibExMastRecvData 関数を実行することで受信データおよび受信データ長を取得することができます。

・1 番目の引数はボードアクセス番号です。
・2 番目の引数はデータを送信してくる機器アドレステーブルを指定します。
・3 番目の引数は受信バッファ長(単位:バイト)です。
　API 呼出し前に受信バッファ長にはあらかじめ受信バッファの長さを格納しておく必要があります。
　受信バッファ長のサイズはデリミタサイズも含めて充分余裕を持って指定してください。
　これは機器から送られてくるデータ長が 10 バイトと判明しており、デリミタに CRLF を指定している場合、受信バッファサイズは最低 12 バイト必要であることを示します。
・4 番目の引数は受信バッファへのポインタです。
・5 番目の引数は同期転送時には WM_NULL または 0 を指定します。非同期転送の場合にはデータ受信完了時にこの引数に設定したメッセージコードがポストされます。

```
char  szData[1024];  // グローバル変数 : 受信バッファ
       ・
int   nAdrsTbl[8];   // 機器アドレステーブル
DWORD dwLen;         // 受信バッファ長を格納、API 呼び出し後には実際の受信データ
長が格納される
       ・
       ・
nAdrsTbl[0] = 1;     // 制御する測定機器のアドレス
```

```
nAdrsTbl[1] = -1;     // アドレステーブルの終端
       ・
       ・
dwLen = 1024;         // 受信バッファのサイズ
PciGpibExMastRecvData(0, nAdrsTbl, &dwLen, szData, 0);
```

関数呼び出し後に変数 dwLen には実際に受信したデータ長が格納されています。

6. シリアル・ポーリング

シリアルポーリングの実行手順は次の通りです。

1.PciGpibExCheckSrq 関数を実行し、機器からの SRQ 信号が有効であるかを確認します。

2.機器からの SRQ 信号が有効であれば、機器からのステータスバイト(STB)を受信するために、

　必ずシリアルポーリング(PciGpibExExecSpoll 関数)を実行します。

7. その他の動作

情報取得、バス・ステータス取得等、他の API は必要に応じて個別に実行します。

```
PciGpibExGetConfig(0, 3, &ulPrm);                      // 送信デリミタ値取得
          ・
          ・
PciGpibExExecSpoll(0, nAdrsTbl, nStbTbl, nStbAdrs);    // シリアル・ポール実行
          ・
          ・
PciGpibExGetStatus(0, &uStatus);                       // バス・ステータス取得
```

コントローラでのみ実行可能なAPIもあります。詳細については『4.1　高機能版DLL』をご参照ください。

8. 終了

インタフェースモジュールを PciGpibExFinishBoard 関数で使用終了します。

必ず本関数を実行して処理を終了してください。

```
PciGpibExFinishBoard(0);
```

2.～7.は初期化、終了の間で必要に応じて繰り返し実行できます(ただし、3.だけは最初の1回のみの実行となります。)

3.4.2 非コントローラとしての実行手順

インタフェースモジュールを非コントローラとして使用する場合の基本的な制御の手順は以下の通りです。（記述例はC言語です）

1. 初期化

インタフェースモジュールを PciGpibExInitBoard 関数で初期化します。このとき、アプリケーションのウィンドウハンドルも一緒に指定します。非同期で送受信を行わない場合や、メッセージポストを行わない場合は WM_NULL または 0 を指定します。

```
PciGpibExInitBoard(0, hWnd);
```

初期化が正常終了後、指定したボードアクセス番号(この場合は 0)で以後の各関数を使用することができます。

2. インタフェースモジュールの設定

PciGpibExSetConfig 関数で、インタフェースモジュールのデリミタ等の各種設定を行います。この関数を実行しない場合は、初期値で設定されます。初期値については、コントロールパネルアプレットで設定した値が使用されます。
また、この関数は初期化(PciGpibExInitBoard 関数)の実行後は何度でも実行できます。

※コントロールパネルアプレットの説明については、README およびユーザーズマニュアルを御参照ください。

```
PciGpibExSetConfig(0, ″/SDELIM=EOI /RDELIM=CRLF+EOI″);
```

送信デリミタとして[EOI のみ]、受信デリミタとして[CRLF+EOI]を設定しました。
以上、非コントローラとしての初期化は完了です。

3. バス・ステータス確認

非コントローラで使用する場合、データの送信/受信を行うためにはインタフェースモジュールがトーカ/リスナの状態になる必要があります。PciGpibExGetStatus 関数にてインタフェースモジュールがどのような状態かを取得することができます。

```
PciGpibExGetStatus(0, &uStatus);
if (uStatus & 0x01000000) {  // インタフェースモジュールはトーカ指定されているか
  // トーカ指定済ならば、データ送信を行う
  DataSend(...);
}
else if (uStatus & 0x02000000) {  // インタフェースモジュールはリスナ指定されているか
    // リスナ指定済ならば、データ受信を行う
    DataRecv(...);
}
```

PciGpibExGetStatus 関数にてインタフェースモジュールの状態を確認後に、データ送信またはデータ受信の処理を実行します。

※ コントローラからデータ送信またはデータ受信を実行した後にマルチラインメッセージ[UNT], [UNL]を送信することで、非コントローラ側のトーカ状態/リスナ状態を完全に解除させせることができます。こうすることで非コントローラ側は次のトーカ指定/リスナ指定を検出することが可能です。

4. 送信

インタフェースモジュールがトーカ指定されている場合には、GP-IB バスへデータ送信を実行することができます。
PciGpibExSlavSendData 関数でデータを送信します。
・1 番目の引数はボードアクセス番号です。
・2 番目の引数は送信データ長(単位:バイト)です。
・3 番目の引数は送信データへのポインタです。
・4 番目の引数は同期転送時には WM_NULL または 0 を指定します。非同期転送の場合にはデータ送信完了時にこの引数に設定したメッセージコードがポストされます。

```
int DataSend(...)
{
          ・
    PciGpibExSlavSendData(0, 10, ″0123456789″, 0);
          ・
}
```

5. 受信

インタフェースモジュールがリスナ指定されている場合には、GP-IB バスからのデータ受信を実行することができます。
PciGpibExSlavRecvData 関数を実行することで受信データおよび受信データ長を取得することができます。

・1 番目の引数はボードアクセス番号です。
・2 番目の引数は受信バッファ長(単位:バイト)です。
　API 呼出し前に受信バッファ長にはあらかじめ受信バッファの長さを格納しておく必要があります。
　受信バッファ長のサイズはデリミタサイズも含めて充分余裕を持って指定してください。
　これは機器から送られてくるデータ長が 10 バイトと判明しており、デリミタに CRLF を指定している場合、受信バッファサイズは最低 12 バイト必要であることを示します。
・3 番目の引数は受信バッファへのポインタです。
・4 番目の引数は同期転送時には WM_NULL または 0 を指定します。非同期転送の場合にはデータ受信完了時にこの引数に設定したメッセージコードがポストされます。

```
char  szData[1024];    // グローバル変数 : 受信バッファ
int DataRecv(...)
{
    DWORD dwLen;  // 受信バッファ長を格納、API 呼び出し後には実際の受信データ長が
                  // 格納される
```

```
        ・
        ・
    dwLen = 1024;        // 受信バッファのサイズ
    PciGpibExSlavRecvData(0, &dwLen, szData, 0);
        ・
        ・
}
```

関数呼び出し後に変数 dwLen には実際に受信したデータ長が格納されています。

6. その他の動作

```
PciGpibExGetConfig(0, 3, &ulPrm);// 送信デリミタ値取得
          ・
          ・
PciGpibExSlavSetSrq(0, 0x5);      // STB 設定、SRQ 信号送出
          ・
          ・
PciGpibExSlavCheckStb(0);         // コントローラよりシリアルポーリングされたかを
確認
```

非コントローラでのみ実行可能なAPIもあります。詳細については『4.1　高機能版DLL』をご参照ください。

7. 終了

インタフェースモジュールを PciGpibExFinishBoard 関数で使用終了します。
必ず本関数を実行して処理を終了してください。

```
PciGpibExFinishBoard(0);
```

2.～6.は初期化、終了の間で必要に応じて繰り返し実行できます。
送信/受信APIを実行する場合には、インタフェースモジュールが送信可能か受信可能かを確認するために「3.バス・ステータス取得」にてインタフェースモジュールの状態をあらかじめ確認しておく必要があります。
その他の状態については、2.～6.のどの時点でもPciGpibExGetStatus関数を呼び出すことが可能です。

3.4.3 Programming Tips

■数秒間隔で連続してデータを送受信する方法について
数秒間隔の連続データ送受信方法については、各言語にて下記の方法が挙げられます。

● Visual C++の場合
1) Win32API の SetTimer 関数を使用し、一定周期のタイマイベントを発生させて、データ送受信を実行。
※C 言語ではマルチメディアタイマを使用することも可能です。マルチメディアタイマの詳
細については書籍もしくは Visual C++の Help などをご参照ください。
2) データ送受信を行う専用のスレッドを作成し、スレッドの中で待ち時間を取りながら、データ送受信を実行。
3) 間隔が多少不正確でもかまわない場合には、機器からの計測完了時の SRQ を利用し、インタフェースモジュール側では PciGpibExSetSrqEvent 関数を使用して SRQ 受信コールバック関数内でデータ送受信を実行。
※SRQ を使用する場合には、シリアルポーリングを実行する必要があります。
4) または、高機能 ActiveX の SRQ イベントを使用し、SRQ イベントプロシージャ内でデータ送受信を実行。
※SRQ を使用する場合には、シリアルポーリングを実行する必要があります。

● Visual Basic の場合
1) タイマコントロールを使用し、一定周期のタイマイベントを発生させて、データ送受信を実行。
2) 間隔が多少不正確でもかまわない場合には、高機能 ActiveX の SRQ イベントを使用し、SRQ イベントプロシージャ内でデータ送受信を実行。
※SRQ を使用する場合には、シリアルポーリングを実行する必要があります。

● Delphi の場合
1) タイマコントロールを使用し、一定周期のタイマイベントを発生させて、データ送受信を実行。
2) 間隔が多少不正確でもかまわない場合には、高機能 ActiveX の SRQ イベントを使用し、SRQ イベントプロシージャ内でデータ送受信を実行。
※SRQ を使用する場合には、シリアルポーリングを実行する必要があります。

SRQの利用方法については、機器のマニュアルもしくは計測機器メーカへお問い合わせください。

■複数台の機器からの SRQ を利用して制御する方法について
各言語毎での SRQ 確認方法と注意事項は次の通りです。

● Visual C++の場合
■パターン 1 PciGpibExCheckSrq 関数を使用して SRQ 受信をチェック
1) PciGpibExCheckSrq 関数で現在の SRQ 受信状況を確認します。

2) SRQ 受信を確認後、複数台の機器に対して PciGpibExExecSpoll 関数を実行(シリアルポーリング)します。

利点: ・ 特にはありません。標準的な手順です。

欠点: ・ メイン処理内で常に SRQ 受信状況を確認しておく必要があります。
→スレッドを使用すれば、メイン処理とは別に、常時確認を実行させることが可能
です。
・ 複数台の機器からほぼ同時(数 ms～数 100ms の間隔)に SRQ が送出された場合、それぞれの機器の SRQ を区別することができません。
常に複数台の機器を指定してシリアルポーリングを実行してください。

■パターン 2 SRQ コールバック関数を使用する

1) あらかじめ、PciGpibExSetSrqEvent 関数で SRQ 受信時のコールバック関数を登録しておきます。
2) 機器からの SRQ を受信、検出すると、ドライバがコールバック関数の呼出しを行います。
3) コールバック関数内で、複数台の機器に対して PciGpibExExecSpoll 関数を実行(シリアルポーリング)します。

利点: ・ SRQ 受信検出の監視をドライバ側で行うため、アプリケーション側で SRQ 受信の確認を行う必要がありません。

欠点: ・ 複数台の機器からほぼ同時(数 ms～数 100ms の間隔)に SRQ が送出された場合、それぞれの機器の SRQ を区別することができません。
そのため、SRQ コールバック関数内の処理としては
a) 1 台のみのシリアルポーリング実行は行わない。
b) SRQ が有効であれば常に複数台の機器をシリアルポーリングする。
ようにしてください。
弊社ドライバのシリアルポーリングは、ポーリングする機器を複数台指定可能です。

■パターン 3 常に複数台の機器に対してシリアルポーリングを実行して SRQ の有無を確認する

1) SRQ 検出の確認を省いて、常に複数台の機器に対して PciGpibExExecSpoll 関数を実行する。

利点: ・ SRQ 受信確認の処理は不要です。
複数台の機器に対して、常にシリアルポーリングを行うことで SRQ の有無とステータスバイト取得の同時を処理することが可能です。
・ スレッドを使用することで、メインの処理とは別にポーリングを実行することが可能です。

欠点: ・ シリアルポーリングの処理は比較的重いため、頻繁に実行させるとパフォーマンスに影響が出る可能性があります。
・ 送受信処理とは別のスレッド内で実行させる場合は、送受信処理とは排他制御を掛けて実行させるようにしてください。

● Visual Basic の場合

■パターン 1 PciGpibExCheckSrq 関数を使用して SRQ 受信をチェック (DLL を直接ご利用の場合)

1) PciGpibExCheckSrq 関数で現在の SRQ 受信状況を確認します。

2) SRQ 受信を確認後、複数台の機器に対して PciGpibExExecSpoll 関数を実行(シリアルポーリング)します。

利点: ・特にありません。一定周期で SRQ 受信の確認を行う場合には、タイマコントロールを使用してください。

欠点: ・複数台の機器からほぼ同時(数 ms～数 100ms の間隔)に SRQ が送出された場合、それぞれの機器の SRQ を区別することができません。
常に複数台の機器を指定してシリアルポーリングを実行してください。

■パターン 2 常に複数台の機器に対してシリアルポーリングを実行して SRQ の有無を確認 (DLL を直接ご利用の場合)

1) SRQ 検出の確認を省いて、常に複数台の機器に対して PciGpibExExecSpoll 関数を実行する。

利点: ・SRQ 受信確認の処理は不要です。
複数台の機器に対して、常にシリアルポーリングを行うことで SRQ の有無とステータスバイト取得の同時を処理することが可能です。
・一定周期でポーリングを実行させる場合には、タイマコントロールを使用してください。

欠点: ・シリアルポーリングの処理は比較的重いため、頻繁に実行させるとパフォーマンスに影響が出る可能性があります。

■パターン 3 高機能版 ActiveX による SRQ イベントを使用

1) IfxGpib::SrqEvent イベントプロシージャ内に ExecSpoll メソッドを記述してシリアルポーリングを実行させる。

利点: ・高機能版 ActiveX を使用することで、Visual C++の SRQ コールバックと同じ形で ActiveX 側が SRQ 受信の検出を行います。
そのため、アプリケーションのプログラミング手順が軽減されます。

欠点: ・Visual Basic ではイベントの処理中に次のイベントを多重に発生させることはできません。
そのため、SRQ イベントプロシージャを実行中に SRQ が発生した場合、次の SRQ イベントが発生しない場合があります。
→上記ケースが発生する場合には「手順 その 4 高機能版 ActiveX を使用して直接
ExecSpoll メソッドを使用」をご利用ください。
・複数台の機器からほぼ同時(数 ms～数 100ms の間隔)に SRQ が送出された場合、それぞれの機器の SRQ を区別することができません。
常に複数台の機器を指定してシリアルポーリングを実行してください。

■パターン 4 高機能版 ActiveX を使用して直接 ExecSpoll メソッドを使用

1) 「手順 その 2」を高機能版 ActiveX で実行させます。
SRQ 検出の確認を省いて、常に複数台の機器に対して ExecSpoll メソッドを実行します。
また、「手順 その 1」の方法についても、高機能版 ActiveX のメソッドを使用して実行す

ることができます。

利点：　・ DLL 直接制御と同様です。

欠点：　・ DLL 直接制御と同様です。

● Delphi の場合

■パターン 1 PciGpibExCheckSrq 関数を使用して SRQ 受信をチェック（DLL を直接ご利用の場合）

1）PciGpibExCheckSrq 関数で現在の SRQ 受信状況を確認します。

2）SRQ 受信を確認後、複数台の機器に対して PciGpibExExecSpoll 関数を実行(シリアルポーリング)します。

利点：　・ 特にはありません。標準的な手順です。

欠点：　・ メイン処理内で常に SRQ 受信状況を確認しておく必要があります。
→スレッド、タイマを使用すれば、メイン処理とは別に、常時確認を実行させること
とが可能です。
・ 複数台の機器からほぼ同時(数 ms～数 100ms の間隔)に SRQ が送出された場合、それぞれの機器の SRQ を区別することができません。
常に複数台の機器を指定してシリアルポーリングを実行してください。

■パターン 2 SRQ コールバック関数を使用する

1）あらかじめ、PciGpibExSetSrqEvent 関数で SRQ 受信時のコールバック関数を登録しておきます。

2）機器からの SRQ を受信、検出すると、ドライバがコールバック関数の呼出しを行います。

3）コールバック関数内で、複数台の機器に対して PciGpibExExecSpoll 関数を実行(シリアルポーリング)します。

利点：　・ SRQ 受信検出の監視をドライバ側で行うため、アプリケーション側で SRQ 受信の確認を行う必要がありません。

欠点：　・ 複数台の機器からほぼ同時(数 ms～数 100ms の間隔)に SRQ が送出された場合、それぞれの機器の SRQ を区別することができません。
そのため、SRQ コールバック関数内の処理としては
a）1 台のみのシリアルポーリング実行は行わない。
b）SRQ が有効であれば常に複数台の機器をシリアルポーリングする。
ようにしてください。
弊社ドライバのシリアルポーリングは、ポーリングする機器を複数台指定可能です。

■パターン 3 常に複数台の機器に対してシリアルポーリングを実行して SRQ の有無を確認する

1）SRQ 検出の確認を省いて、常に複数台の機器に対して PciGpibExExecSpoll 関数を実行する。

利点：　・ SRQ 受信検出の監視をドライバ側で行うため、アプリケーション側で SRQ 受信の確認を行う必要がありません。

欠点：　・ メイン処理内で常に SRQ 受信状況を確認しておく必要があります。
→スレッド、タイマを使用すれば、メイン処理とは別に、常時確認を実行させること

とが可能です。

・ 複数台の機器からほぼ同時(数 ms～数 100ms の間隔)に SRQ が送出された場合、それぞれの機器の SRQ を区別することができません。
常に複数台の機器を指定してシリアルポーリングを実行してください。

■パターン 4 高機能版 ActiveX コントロールを使用する

1) Delphi 上で高機能版 ActiveX コントロールを使用することで、SRQ イベントを利用することが可能です。

利点: ・「●Visual Basic の場合」を参照ください。

欠点: ・「●Visual Basic の場合」を参照ください。

3.5 標準版DLL実行手順

GP-IB製品を複数枚使用する場合は、
・Compact PCI, PCI製品の場合、インタフェースモジュール上のロータリースイッチRSW1の設定値
・CardBus製品の場合、CardBus IDの設定値
がGP-IB製品同士で重複しないように設定してからスロットに実装し、システムを起動して下さい。GP-IB製品が複数存在する場合、制御対象を一意に識別するための番号となります。重複していた場合、本ソフトウェアは正常に動作いたしません。

基本的な制御の手順は以下の通りです（記述例はC言語です）。

1. 初期化

インタフェースモジュールを GpibOpen 関数で初期化します。ボード番号を指定します。

```
GpibOpen( ulBoardNo );
```

初期化が正常終了後、指定したボード番号で以後の各関数を使用することができます。

2. インタフェースモジュールの設定

GpibSetConfig 関数で、デリミタ等のインタフェースモジュールの設定を行います。この関数を実行しない場合は、初期値で設定されます。初期値については、コントロールパネルで設定した値が使用されます。また、この関数は初期化の実行後は何度でも実行できます。

```
GpibSetConfig( ulBoardNo, ″/SDELIM=EOI /RDELIM=CRLF+EOI″ );
```

3. GP-IB インタフェースの初期化

GpibSetIfc 関数を使用して GP-IB バスの初期化を行います。続いて、GpibSetRen 関数を使用して REN 信号を有効にします。

```
GpibSetIfc( ulBoardNo );
GpibSetRen( ulBoardNo );
```

4. 送信

送りたいデータを GpibSend 関数で指定した機器に対して送信します。
機器アドレステーブルの終端には必ず -1 を代入してください。

```
int nAdrsTbl[8];
      ・
nAdrsTbl[0] = 1;     // 制御する測定機器のアドレス
nAdrsTbl[1] = -1;    // アドレステーブルの終端
      ・
GpibSend( ulBoardNo, nAdrsTbl, 10, ″0123456789″ );
```

5. 受信

受信データおよび受信データ長を GpibReceive 関数で指定した機器から受信します。
受信バッファサイズの指定はデリミタサイズも含めて充分余裕を持って指定してください。
これは、機器から送られてくるデータ長が 10 バイトと判明しており、デリミタに CRLF を指定している場合、受信バッファサイズは最低 12 バイト必要であることを示します。

```
int nAdrsTbl[8];
DWORD dwLen;
char szData[1024];
      ・
nAdrsTbl[0] = 1;    // 制御する測定機器のアドレス
nAdrsTbl[1] = -1;  // アドレステーブルの終端
      ・
dwLen = 1024;      // 受信バッファのサイズ

GpibReceive( ulBoardNo, nAdrsTbl, &dwLen, szData );
```

関数呼び出し後に、変数 dwLen には実際に受信したデータ長が格納されています。

6. シリアル・ポール実行

```
GpibExecSpoll( ulBoardNo, nAdrsTbl, nStbTbl, nStbAdrs );  // シリアル・ポール実行
```

7. SRQ イベントコールバックの登録/解除

SRQ イベントが発生した時に、登録した関数を実行させることができます。
これをコールバックイベント機能と呼びます。
GpibSetSrqEvent 関数にてイベントが発生した時にコールされる関数を登録します。
イベントが発生すると、設定した関数がコールされます。
GpibWaitSrqEvent 関数にてタイムアウトを設定し、イベントを待つことができます。
イベントの処理が必要なくなった場合は、GpibKillSrqEvent 関数でコールバック関数の登録を解除します。

8. 終了処理

インタフェースモジュールを GpibClose 関数で使用終了します。
必ず本関数を実行して処理を終了してください。

```
GpibClose( ulBoardNo );
```

2.～6. は初期化、終了の間で繰り返し実行できます(ただし、3. だけは最初の 1 回のみの実行となります)。

3.6 機器アドレステーブルについて

機器アドレステーブルには、1次、2次アドレスを混在して複数台の機器を指定することができます。コントローラ時、機器の指定が必要な関数にて機器アドレステーブルを引数とする場合、以下のようにint型（整数型）の配列に機器のアドレスを設定して使用します。

3.6.1 データ送信時

弊社GP-IBインタフェースからデータを送信し、指定機器で受信します。
アドレスとして指定するのは全て受信先機器のアドレスとなります。

■1 次アドレスのみの場合（1 台だけを指定する場合）

配列Index	格納値
0	1次アドレス
1	終端（-1）

【例】

受信先機器のアドレスが 1 次アドレス=2 の場合

●C 言語

```
int nAdrs[2] = { 2, -1 };
```

●Visual Basic

```
Dim nAdrs(1) As Long
nAdrs(0) = 2
nAdrs(1) = -1
```

●Delphi

```
var
  nAdrs[0..2] of Integer;

nAdrs[0] := 2;
nAdrs[1] := -1;
```

■2 次アドレスを設定した場合（1 台だけを指定する場合）

配列Index	格納値
0	1次アドレス
1	2次アドレス
2	終端（-1）

【例】

受信先機器のアドレスが 1 次アドレス=2、2 次アドレス=96 の場合

●C 言語

```
int nAdrs[3] = { 2, 96, -1 };
```

●Visual Basic

```
Dim nAdrs(2) As Long
nAdrs(0) = 2
nAdrs(1) = 96
nAdrs(2) = -1
```

●Delphi

```
var
  nAdrs[0..3] of Integer;

nAdrs[0] := 2;
nAdrs[1] := 96;
nAdrs[2] := -1;
```

■1 次アドレスのみの場合（複数台を指定する場合）

配列Index	格納値
0	1次アドレス
1	1次アドレス
…	…
n-1	1次アドレス
n	終端（-1）

【例】

１台目の受信先機器のアドレスが 1 次アドレス=2、2 台目の受信先機器のアドレスが 1 次アドレス=7 の場合（本ドライバが送信したデータは 2 台の機器とも同じデータを同時に受信します）

●C 言語

```
int nAdrs[3] = { 2, 7, -1 };
```

●Visual Basic

```
Dim nAdrs(2) As Long
nAdrs(0) = 2
nAdrs(1) = 7
nAdrs(2) = -1
```

●Delphi

```
var
  nAdrs[0..3] of Integer;

nAdrs[0] := 2;
nAdrs[1] := 7;
nAdrs[2] := -1;
```

■2 次アドレスを設定した場合（複数台を指定する場合）

配列Index	格納値
0	1次アドレス
1	2次アドレス
2	1次アドレス
3	2次アドレス
…	…
n-2	1次アドレス
n-1	2次アドレス
n	終端（-1）

【例】

1 台目の受信先機器のアドレスが 1 次アドレス=2、2 次アドレス=96、2 台目の受信先機器のアドレスが 1 次アドレス=7、2 次アドレス=97 の場合（本ドライバが送信したデータは 2 台の機器とも同じデータを同時に受信します）

●C 言語

```
int nAdrs[5] = { 2, 96, 7, 97, -1 };
```

●Visual Basic

```
Dim nAdrs(4) As Long
nAdrs(0) = 2
nAdrs(1) = 96
nAdrs(2) = 7
nAdrs(3) = 97
nAdrs(4) = -1
```

●Delphi

```
var
  nAdrs[0..5] of Integer;

nAdrs[0] := 2;
nAdrs[1] := 96;
nAdrs[2] := 7;
nAdrs[3] := 97;
nAdrs[4] := -1;
```

3.6.2 データ受信時

配列の先頭位置に記述されたアドレスの機器からデータを送信し、弊社GP-IBインタフェース（および、配列に続けて記述されたアドレスの機器）で受信します。
配列の先頭位置に記述されたアドレスが、送信機器アドレスとなります。
この時、他に受信させる機器を指定する場合には、続いて機器アドレスを記述することにより設定します。

■1 次アドレスのみの場合（1 台だけを指定する場合）

配列Index	格納値
0	1次アドレス（送信機器）
1	終端（-1）

【例】

送信元機器のアドレスが 1 次アドレス=2 の場合

●C 言語

```
int nAdrs[2] = { 2, -1 };
```

●Visual Basic

```
Dim nAdrs(1) As Long
nAdrs(0) = 2
nAdrs(1) = -1
```

●Delphi

```
var
  nAdrs[0..2] of Integer;

nAdrs[0] := 2;
nAdrs[1] := -1;
```

■2 次アドレスを設定した場合（1 台だけを指定する場合）

配列Index	格納値
0	1次アドレス（送信機器）
1	2次アドレス（送信機器）
2	終端（-1）

【例】

送信元機器のアドレスが 1 次アドレス=2、2 次アドレス=96 の場合

●C 言語

```
int nAdrs[3] = { 2, 96, -1 };
```

●Visual Basic

```
Dim nAdrs(2) As Long
nAdrs(0) = 2
nAdrs(1) = 96
nAdrs(2) = -1
```

●Delphi

```
var
  nAdrs[0..3] of Integer;

nAdrs[0] := 2;
nAdrs[1] := 96;
nAdrs[2] := -1;
```

■1 次アドレスのみの場合（他の受信機器も指定する場合）

配列Index	格納値
0	1次アドレス(送信機器)
1	1次アドレス(受信機器)
…	…
n-1	1次アドレス(受信機器)
n	終端（-1）

【例】

送信元機器のアドレスが 1 次アドレス=2、受信先機器のアドレスが 1 次アドレス=7 の場合
（1 台目の機器が送信したデータを本ドライバと受信先の機器とが同時に受信します）

●C 言語

```
int nAdrs[3] = { 2, 7, -1 };
```

●Visual Basic

```
Dim nAdrs(2) As Long
nAdrs(0) = 2
nAdrs(1) = 7
nAdrs(2) = -1
```

●Delphi

```
var
  nAdrs[0..3] of Integer;

nAdrs[0] := 2;
nAdrs[1] := 7;
nAdrs[2] := -1;
```

■2 次アドレスを設定した場合（他の受信機器も指定する場合）

配列Index	格納値
0	1次アドレス(送信機器)
1	2次アドレス(送信機器)
2	1次アドレス(受信機器)
3	2次アドレス(受信機器)
…	…
n-2	1次アドレス(受信機器)
n-1	2次アドレス(受信機器)
n	終端（-1）

【例】

送信元機器のアドレスが 1 次アドレス=2、2 次アドレス=96、受信先機器のアドレスが 1 次アドレス=7、2 次アドレス=97 の場合（1 台目の機器が送信したデータを本ドライバと受信先の機器とが同時に受信します）

●C 言語

```
int nAdrs[5] = { 2, 96, 7, 97, -1 };
```

●Visual Basic

```
Dim nAdrs(4) As Long
nAdrs(0) = 2
nAdrs(1) = 96
nAdrs(2) = 7
nAdrs(3) = 97
nAdrs(4) = -1
```

●Delphi

```
var
  nAdrs[0..5] of Integer;

nAdrs[0] := 2;
nAdrs[1] := 96;
nAdrs[2] := 7;
nAdrs[3] := 97;
nAdrs[4] := -1;
```

第4章 リファレンス

4.1 高機能版DLL

4.1.1 関数一覧

No	関数名	機能
1	PciGpibExInitBoard	インタフェースモジュールの初期化を行います。
2	PciGpibExFinishBoard	インタフェースモジュールの使用を終了します。
3	PciGpibExGetInfo	インタフェースモジュールの情報を取得します。
4	PciGpibExSetConfig	ドライバの動作パラメータを設定／変更します。
5	PciGpibExGetConfig	ドライバの動作パラメータを取得します。
6	PciGpibExSetIfc	IFCの送出を行います。 (System Controller のみ)
7	PciGpibExSetRen	RENラインをセットします。 (System Controller のみ)
8	PciGpibExResetRen	RENラインをリセットします。 (System Controller のみ)
9	PciGpibExSetRemote	指定した機器をリモートモードに設定します。 (System Controller のみ)
10	PciGpibExExecTrigger	指定した機器に対してトリガを行います。 (CIC – Master のみ)
11	PciGpibExExecDevClear	全ての機器に対してデバイスクリアを行います。 (CIC – Master のみ)
12	PciGpibExExecSdc	指定した機器に対してデバイスクリアを行います。 (CIC – Master のみ)
13	PciGpibExSetLocal	指定した機器をローカルモードに設定します。 (CIC – Master のみ)
14	PciGpibExSetLlo	全ての機器をローカルロックアウト状態に設定します。 (CIC – Master のみ)
15	PciGpibExSetRwls	指定した機器をリモートロックアウト状態に設定します。 (System Controller のみ)
16	PciGpibExExecPassCtrl	指定した機器に対してパス・コントロールを行います。 (CIC – Master のみ)
17	PciGpibExExecFindListener	バス上に接続されているリスナを探します。 (CIC – Master のみ)
18	PciGpibExExecDevReset	指定した機器を完全にリセットします。 (System Controller のみ)
19	PciGpibExReSysCtrl	システムコントローラの要求または解除を行います。 (System Controller のみ)
20	PciGpibExGoStandby	コマンドモードからデータモードへ遷移させます。 (CIC – Master のみ)
21	PciGpibExGoActCtrller	データモードからコマンドモードへ遷移させます。 (CIC – Master のみ)
22	PciGpibExExecSpoll	指定した機器をシリアルポーリングします。 (CIC – Master のみ)

No	関数名	機能
23	PciGpibExCheckSrq	SRQ受信の有効/無効を確認します。 (CIC – Master のみ)
24	PciGpibExClearSrq	SRQ受信フラグのクリアを行います。 (CIC – Master のみ)
25	PciGpibExEnableSrq	SRQ受信の許可を行います。 (CIC – Master のみ)
26	PciGpibExDisableSrq	SRQ受信の禁止を行います。 (CIC – Master のみ)
27	PciGpibExExecPpoll	パラレルポーリングを行います。 (CIC – Master のみ)
28	PciGpibExCfgPpoll	パラレルポール応答条件を設定します。 (CIC - Master のみ)
29	PciGpibExUnCfgPpoll	パラレルポール応答条件を解除します。 (CIC - Master のみ)
30	PciGpibExWriteBusCmd	バスコマンドの送出を行います。 (CIC - Master のみ)
31	PciGpibExSetSignal	事象変化検出条件の設定を行います。
32	PciGpibExWaitSignal	事象変化検出を待ちます。
33	PciGpibExGetStatus	現在のバス・ステータスを取得します。
34	PciGpibExClrStatus	現在のバス・ステータスのクリアを行います。
35	PciGpibExMastSendData	バスにデータの送信を行います。 (CIC - Master のみ)
36	PciGpibExMastRecvData	バスからデータの受信を行います。 (CIC - Master のみ)
37	PciGpibExMastSendFile	ファイルから読み込んだデータをバスに送信します。 (CIC - Master のみ)
38	PciGpibExMastRecvFile	バスから受信したデータをファイルに書き込みます。 (CIC - Master のみ)
39	PciGpibExSlavSendData	バスにデータの送信を行います。 (CIC - Master / Not CIC - Slave)
40	PciGpibExSlavRecvData	バスからデータの受信を行います。 (CIC - Master / Not CIC - Slave)
41	PciGpibExSlavSendFile	ファイルから読み込んだデータをバスに送信します。 (CIC - Master / Not CIC - Slave)
42	PciGpibExSlavRecvFile	バスから受信したデータをファイルに書き込みます。 (CIC - Master / Not CIC - Slave)
43	PciGpibExSlavCheckStb	コントローラからシリアルポーリングされたかを確認します。
44	PciGpibExSlavSetSrq	SRQの送出(サービス要求)を行います。
45	PciGpibExSlavSetIst	パラレル・ポール応答フラグを設定します。
46	PciGpibExSlavSetPp2	パラレル・ポール応答モード(pp2)を設定します。
47	PciGpibExWaitTimer	指定した時間待ちます。
48	PciGpibExStartTimer	汎用タイマのスタートを行います。
49	PciGpibExClearTimer	汎用タイマのカウンタ値のクリアを行います。
50	PciGpibExReadTimer	汎用タイマのカウンタ値の読み出しを行います。
51	PciGpibExStopTimer	汎用タイマの停止を行います。
52	PciGpibExSetSrqEvent	SRQコールバックイベントを登録します。 (CIC - Master のみ)

No	関数名	機能
53	PciGpibExWaitSrqEvent	SRQコールバックイベントを待ちます。 (CIC - Master のみ)
54	PciGpibExKillSrqEvent	SRQコールバックイベントの登録を解除します。 (CIC - Master のみ)
57	PciGpibExMastSendDelay	バスコマンドの間隔を指定してバスにデータの送信を行います。 (CIC - Master のみ)
58	PciGpibExMastRecvDelay	バスコマンドの間隔を指定してバスからデータの受信を行います。(CIC - Master のみ)

4.1.2 関数個別説明

1. PciGpibExInitBoard

【機能】

インタフェースモジュールの初期化を行います。

【書式】

●C 言語

```
int PciGpibExInitBoard (
  int        BoardNo,
  HWND       Wnd
);
```

●Visual Basic

```
Declare Function PciGpibExInitBoard Lib "gpc4304.dll"( _
  ByVal  BoardNo    As Long,    _
  ByVal  Wnd        As Long     _
) As Long
```

●Delphi

```
function PciGpibExInitBoard (
     BoardNo   : Integer;
     Wnd       : Cardinal
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。
インタフェースモジュール上のロータリスイッチの値がボードアクセス番号となります。
CardBus シリーズにつきましては、CardBus ID 設定ユーティリティで設定した ID がボードアクセス番号となります。
ソルコン Classembly Devices®、I/O 付きタッチパネル Classembly Devices®の GP-IB モデルをお使いの場合は、0 を指定してください。
※複数枚使用時、ボードアクセス番号はそれぞれのインタフェースモジュールで重複しない番号を割り振ってください。

Wnd

アプリケーションのウィンドウハンドルを指定します。

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

あらかじめコントロールパネルにて、ボードアクセス番号に対応するデバイスの初期設定しておく必要があります。

【使用例】

●C 言語

```
int Ret;
Ret = PciGpibExInitBoard( 0, hWnd );
```

●Visual Basic

```
Dim Ret As Long
Ret = PciGpibExInitBoard( 0, 0 )  ' Visual Basic ではウィンドウハンドルに
                                  ' 0 を指定してください
```

●Delphi

```
var
  Ret : Integer;

Ret := PciGpibExInitBoard( 0, TfrmInitBoard.HANDLE);
```

インタフェースモジュールを初期化します。

2. PciGpibExFinishBoard

【機能】

インタフェースモジュールの使用を終了します。

【書式】

●C 言語

```
int PciGpibExFinishBoard (
   int       BoardNo
);
```

●Visual Basic

```
Declare Function PciGpibExFinishBoard Lib "gpc4304.dll"( _
   ByVal   BoardNo          As Long    _
) As Long
```

●Delphi

```
function PciGpibExFinishBoard (
      BoardNo    : Integer
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【使用例】

●C 言語

```
int Ret;
Ret = PciGpibExFinishBoard( 0 );
```

●Visual Basic

```
Dim Ret As Long
Ret = PciGpibExFinishBoard( 0 )
```

●Delphi

```
var
  Ret : Integer;

Ret := PciGpibExFinishBoard( 0 );
```

インタフェースモジュールの使用を終了します。

3. PciGpibExGetInfo

【機能】

設定情報を取得します。

【書式】

●C 言語

```
int PciGpibExGetInfo(
    int          BoardNo,
    int          PrmNo,
    ULONG*       GetPrm
);
```

●Visual Basic

```
Declare Function PciGpibExGetInfo Lib "gpc4304.dll"( _
    ByVal   BoardNo    As Long,          _
    ByVal   PrmNo      As Long,          _
    ByRef   GetPrm     As Long           _
) As Long
```

●Delphi

```
function PciGpibExGetInfo (
        BoardNo     : Integer;
        PrmNo       : Integer;
    var GetPrm      : Integer
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

PrmNo

インタフェースモジュールの設定情報を取得するためのパラメータ番号を指定します。

パラメータ番号	説明
0	ベースアドレス インタフェースモジュールのベースアドレスを取得します。
1	割り込み番号 インタフェースモジュールの割り込み番号を取得します。
2	動作モード インタフェースモジュールの初期化時の動作モードを取得します。 0: コントローラ 1: 非コントローラ

※ソルコン Classembly Devices®、I/O 付きタッチパネル Classembly Devices®の GP-IB モデルをお使いの場合は、パラメータ番号 2 以外は取得できません。

GetPrm

情報を格納する変数へのポインタ（参照渡し）を指定してください。

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【使用例】

●C 言語

```
int Ret;
ULONG GetPrm;
Ret = PciGpibExGetInfo( 0, 2, &GetPrm );
```

●Visual Basic

```
Dim Ret As Long
Dim GetPrm As Long
Ret = PciGpibExGetInfo( 0, 2, GetPrm )
```

●Delphi

```
var
  Ret : Integer;
  GetRrm : Integer;

Ret := PciGpibExGetInfo( 0, 2, GetPrm );
```

ボードアクセス番号 0 のインタフェースモジュールの動作モードを取得します。

4. PciGpibExSetConfig

【機能】

ドライバの動作パラメータを設定／変更します。

【書式】

●C 言語

```
int PciGpibExSetConfig (
    int       BoardNo,
    char*     Config
);
```

●Visual Basic

```
Declare Function PciGpibExSetConfig Lib "gpc4304.dll"( _
    ByVal   BoardNo    As Long,     _
    ByVal   Config     As String    _
) As Long
```

●Delphi

```
function PciGpibExSetConfig (
        BoardNo   : Integer;
        Config    : String
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

Config

設定に関する各種情報を含んだ文字列を指定してください。
指定のなかった項目に対しては、初期値が設定されます。
設定情報の各パラメータを以下に示します。各パラメータは、スペースで区切ってください。各パラメータ文字列の中にはスペースを入れないようにしてください。
文字列の最後に NULL 文字を入れてください。

項目	文字列	内容
送受信タイムアウト	"/TMO="	送受信時に使用するタイムアウト時間(単位：100ms)を 1～65535 で設定します。 デフォルトは、コントロールパネルアプレットでの設定値が使用されます。
STB 応答時間	"/SRT="	シリアルポール時のステータスバイト応答時間(単位：100ms)を 1～65535 で設定します。 デフォルトは、コントロールパネルアプレットでの設定値が使用されます。
事象変化検出タイムアウト	"/STM="	事象変化検出(PciGpibExSetSignal 関数、PciGpibExWaitSignal 関数)でのタイムアウト時間(単位：100ms)を設定します。

<table>
<tr><th>項目</th><th>文字列</th><th colspan="2">内容</th></tr>
<tr><td></td><td></td><td colspan="2">0:タイムアウトなし
1～65535:タイムアウトあり
デフォルトは、コントロールパネルアプレットでの設定値が使用されます。</td></tr>
<tr><td rowspan="11">送信デリミタ</td><td rowspan="11">"/SDELIM="</td><td colspan="2">送信時に使用するデリミタを設定します。
デフォルトは、コントロールパネルアプレットでの設定値が使用されます。</td></tr>
<tr><td>"NO"</td><td>デリミタ無し</td></tr>
<tr><td>"EOI"</td><td>EOI</td></tr>
<tr><td>"CR"</td><td>CR</td></tr>
<tr><td>"CR+EOI"</td><td>CR+EOI</td></tr>
<tr><td>"LF"</td><td>LF</td></tr>
<tr><td>"LF+EOI"</td><td>LF+EOI</td></tr>
<tr><td>"CRLF"</td><td>CRLF</td></tr>
<tr><td>"CRLF+EOI"</td><td>CRLF+EOI</td></tr>
<tr><td>"NULL"</td><td>NULL (0x00)</td></tr>
<tr><td>"!x"</td><td>感嘆符に続く任意の 1 文字をデリミタとする</td></tr>
<tr><td rowspan="11">受信デリミタ</td><td rowspan="11">"/RDELIM="</td><td colspan="2">受信時に使用するデリミタを設定します。
デフォルトは、コントロールパネルアプレットでの設定値が使用されます。</td></tr>
<tr><td>"NO"</td><td>デリミタ無し</td></tr>
<tr><td>"EOI"</td><td>EOI</td></tr>
<tr><td>"CR"</td><td>CR</td></tr>
<tr><td>"CR+EOI"</td><td>CR+EOI</td></tr>
<tr><td>"LF"</td><td>LF</td></tr>
<tr><td>"LF+EOI"</td><td>LF+EOI</td></tr>
<tr><td>"CRLF"</td><td>CRLF</td></tr>
<tr><td>"CRLF+EOI"</td><td>CRLF+EOI</td></tr>
<tr><td>"NULL"</td><td>NULL (0x00)</td></tr>
<tr><td>"!x"</td><td>感嘆符に続く任意の 1 文字をデリミタとする</td></tr>
<tr><td rowspan="3">非同期入出力</td><td rowspan="3">"/ASYNC="</td><td colspan="2">非同期入出力の設定をします。</td></tr>
<tr><td>"ON"</td><td>データの送受信において、非同期による入出力を行います。</td></tr>
<tr><td>"OFF"</td><td>データの送受信において、同期しての入出力を行います。(デフォルト)</td></tr>
<tr><td>1 次アドレス</td><td>"/MA="</td><td colspan="2">1 次アドレスを設定します。範囲 0～30
デフォルトは、コントロールパネルアプレットでの設定値が使用されます。</td></tr>
<tr><td>2 次アドレス</td><td>"/SA="</td><td colspan="2">2 次アドレスを設定します。範囲：96～126
96～126 以外の値を設定すると 2 次アドレスは無効となります。
デフォルトは、コントロールパネルアプレットでの設定値が使用されます。</td></tr>
<tr><td rowspan="4">IEEE-488 バスハンドシェイクタイミング</td><td rowspan="4">"/HSTMG="</td><td colspan="2">IEEE-488 ソースハンドシェイク T1 タイミングの設定を行います。
デフォルトは、コントロールパネルアプレットでの設定値が使用されます。</td></tr>
<tr><td>0</td><td>T1 に正常タイミング(2μs)</td></tr>
<tr><td>1</td><td>T1 に高速タイミング(500ns)</td></tr>
<tr><td>2</td><td>T1 に超高速タイミング(350ns)</td></tr>
</table>

項目	文字列	内容	
パラレルポール実行時間	″/PPETM=″	パラレルポール実行時間 T6 の設定を行います。 デフォルトは、コントロールパネルアプレットでの設定値が使用されます。	
		0	T6 を 2μs(デフォルト)
		1	T6 を 10μs
NRFD 信号ラインアサート待ち[有効／無効]	″/NRFDWAIT=″	コントローラデータ送信の最終データ送信後に GP-IB バスの NRFD 信号ラインがアサートされるまで「待つ」か「待たない」かの設定を行います。 「待ち時間」には「データ転送タイムアウト」の時間が適用されます。	
		″ON″	NRFD 信号ラインがアサートされるまで待ちます。
		″OFF″	NRFD 信号ラインのアサート待ちは無効となります。(デフォルト)
バスコマンド送信タイムアウト	″/CMDTMO=″	GP-IB バスコマンド送信時に使用するタイムアウト時間(単位：100ms)を 1～65536 で設定します。 デフォルトは、コントロールパネルアプレットでの設定値が使用されます。	
コントローラ	″/CONTROLLER=″	予約です。 動作モードはコントロールパネルアプレットで設定してください。	
		″ON″	予約です。
		″OFF″	予約です。

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

各設定情報のデフォルトは、コントロールパネルアプレットで設定した情報が使用されます。

【使用例】

●C 言語

```
int Ret;
Ret = PciGpibExSetConfig( 0, ″/SDELIM=CRLF /RDELIM=EOI″ );
```

●Visual Basic

```
Dim Ret As Long
Ret = PciGpibExSetConfig( 0, ″/SDELIM=CRLF /RDELIM=EOI″ )
```

●Delphi

```
var
  Ret : Integer;

Ret := PciGpibExSetConfig( 0, '/SDELIM=CRLF /RDELIM=EOI' );
```

ボードアクセス番号 0 のインタフェースモジュールを送信デリミタ CRLF、受信デリミタ EOI として、設定します。

5. PciGpibExGetConfig

【機能】

ドライバの動作パラメータを設定／変更します。

【書式】

●C 言語

```
int PciGpibExGetConfig (
    int         BoardNo,
    int         PrmNo,
    ULONG*      GetPrm
);
```

●Visual Basic

```
Declare Function PciGpibExSetConfig Lib "gpc4304.dll"( _
    ByVal   BoardNo    As Long,    _
    ByVal   PrmNo      As Long,    _
    ByRef   GetPrm     As Long     _
) As Long
```

●Delphi

```
function PciGpibExSetConfig (
        BoardNo    : Integer;
        PrmNo      : Integer;
        GetPrm     : Cardinal
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

PrmNo

動作パラメータを取得するためのパラメータ番号を指定します。

PrmNo	項目	内容	
0	送受信タイムアウト	送受信時に使用するタイムアウト時間(単位：100ms)を取得します。	
1	STB 応答時間	シリアルポール時のステータスバイト応答時間(単位：100ms) を取得します。	
2	事象変化検出タイムアウト	事象変化検出でのタイムアウト時間(単位：100ms)を取得します。	
3	送信デリミタ	送信時に使用するデリミタを取得します。	
		0	デリミタ無し
		1	EOI
		2	CR

		3	CR+EOI
		4	LF
		5	LF+EOI
		6	CRLF
		7	CRLF+EOI
		8	NULL（0x00）
		9	感嘆符に続く任意の 1 文字をデリミタとする
4	受信デリミタ	受信時に使用するデリミタを取得します。	
		0	デリミタ無し
		1	EOI
		2	CR
		3	CR+EOI
		4	LF
		5	LF+EOI
		6	CRLF
		7	CRLF+EOI
		8	NULL（0x00）
		9	感嘆符に続く任意の 1 文字をデリミタとする
5	非同期入出力	非同期入出力の取得をします。	
		0	データの送受信において、同期しての入出力を行います。
		1	データの送受信において、非同期による入出力を行います。
6	1 次アドレス	1 次アドレスを取得します。	
7	2 次アドレス	2 次アドレスを取得します。 255 のとき 2 次アドレスは無効です。	
8	IEEE-488 バスハンドシェイクタイミング	IEEE-488 ソースハンドシェイク T1 タイミングの取得を行います。	
		0	T1 に正常タイミング(2μs)
		1	T1 に高速タイミング(500ns)
		2	T1 に超高速タイミング(350ns)
9	パラレルポール実行時間	パラレルポール実行時間 T6 の取得を行います。	
		0	T6 を 2μs(デフォルト)
		1	T6 を 10μs

GetPrm

パラメータを格納する変数へのポインタ（参照渡し）を指定します。

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【使用例】

●C 言語

```
int Ret;
ULONG GetPrm;
Ret = PciGpibExGetConfig( 0, 0, &GetPrm );
```

●Visual Basic

```
Dim Ret As Long
Dim GetRrm As Long
Ret = PciGpibExGetConfig( 0, 0, GetPrm )
```

●Delphi

```
var
  Ret : Integer;
  GetPrm : Cardinal;

Ret := PciGpibExGetConfig( 0, 0, GetPrm );
```

ボードアクセス番号 0 のインタフェースモジュールの現在の送受信タイムアウト時間を取得します。

6. PciGpibExSetIfc

【機能】

IFC 信号の送出を行います。

【書式】

●C 言語

```
int PciGpibExSetIfc(
    int        BoardNo,
    int        Time
);
```

●Visual Basic

```
Declare Function PciGpibExSetIfc Lib "gpc4304.dll"( _
    ByVal   BoardNo    As Long,             _
    ByVal   Time       As Long              _
) As Long
```

●Delphi

```
function PciGpibExSetIfc (
        BoardNo    : Integer;
        Time       : Integer
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

Time

IFC 送出時間(1～255)を指定します。

100μs 単位で IFC 送出時間を設定します。

【戻り値】

正常終了した場合は、0 が返されます。

0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

・　本関数はシステムコントローラでのみ使用できます。

【使用例】

●C 言語

```
int Ret;
Ret = PciGpibExSetIfc( 0, 1 );
```

●Visual Basic

```
Dim Ret As Long
Ret = PciGpibExSetIfc( 0, 1 )
```

●Delphi

```
var
  Ret : Integer;

Ret := PciGpibExSetIfc( 0, 1 );
```

ボードアクセス番号 0 のインタフェースモジュールから IFC 送出を行います。

7. PciGpibExSetRen

【機能】

REN 信号を有効にします。

【書式】

●C 言語

```
int PciGpibExSetRen (
    int   BoardNo
);
```

●Visual Basic

```
Declare Function PciGpibExSetRen Lib "gpc4304.dll"( _
    ByVal   BoardNo     As Long   _
) As Long
```

●Delphi

```
function PciGpibExSetRen (
        BoardNo   : Integer
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

・ 本関数はシステムコントローラでのみ使用できます。
・ システムコントローラとして IFC 送出に引き続いて REN 信号を有効にすることで機器をリモート状態に設定します。

【使用例】

●C 言語

```
int Ret;
Ret = PciGpibExSetRen( 0 );
```

●Visual Basic

```
Dim Ret As Long
Ret = PciGpibExSetRen( 0 )
```

●Delphi

```
var
  Ret : Integer;

Ret := PciGpibExSetRen( 0 );
```

ボードアクセス番号０のインタフェースモジュールの REN 信号を有効にします。

8. PciGpibExResetRen

【機能】

REN 信号を無効にします。

【書式】

●C 言語

```
int PciGpibExResetRen(
    int         BoardNo
);
```

●Visual Basic

```
Declare Function PciGpibExResetRen Lib "gpc4304.dll"( _
    ByVal   BoardNo     As Long     _
) As Long
```

●Delphi

```
function PciGpibExResetRen (
        BoardNo     : Integer
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

・ 本関数はシステムコントローラでのみ使用できます。
・ すべてのデバイスをローカル状態(フロント・パネル制御可)の状態にします。

【使用例】

●C 言語

```
int Ret;
Ret = PciGpibExResetRen( 0 );
```

●Visual Basic

```
Dim Ret As Long
Ret = PciGpibExResetRen( 0 )
```

●Delphi

```
var
  Ret : Integer;

Ret := PciGpibExResetRen( 0 );
```

REN 信号を無効にします。

9. PciGpibExSetRemote

【機能】

指定した機器をリモートモードにします。

【書式】

●C 言語

```
int PciGpibExSetRemote (
   int        BoardNo,
   int*       AdrsTbl
);
```

●Visual Basic

```
Declare Function PciGpibExSetRemote Lib "gpc4304.dll"( _
   ByVal    BoardNo        As Long,        _
   ByRef    AdrsTbl        As Long         _
) As Long
```

●Delphi

```
function PciGpibExSetRemote (
      BoardNo  : Integer;
      AdrsTbl  : Pointer
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

AdrsTbl

機器のアドレステーブルを指定します。
リモートモードに設定する機器のアドレステーブルへのポインタ（参照渡し）を指定します。アドレステーブルへの終端には必ず-1 を設定します。

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

- 本関数はシステムコントローラでのみ使用できます。
- 機器を GP-IB による制御可能状態とし，フロントパネルからの制御を禁止します。

【使用例】

●C 言語

```
int Ret;
int AdrsTbl[2];

AdrsTbl[0] = 2;
AdrsTbl[1] = -1;
Ret = PciGpibExSetRemote( 0, AdrsTbl );
```

●Visual Basic

```
Dim Ret As Long
Dim AdrsTbl(1) As Long

AdrsTbl(0) = 2
AdrsTbl(1) = -1
Ret = PciGpibExSetRemote( 0, AdrsTbl(0) )
```

●Delphi

```
var
  Ret : Integer;
  AdrsTbl : Array[0..4] of Integer;

AdrsTbl[0] := 2;
AdrsTbl[1] := -1;
Ret := PciGpibExSetRemote ( 0, @AdrsTbl[0] );
```

ボードアクセス番号 0 のインタフェースモジュールに対して AdrsTbl に記述した機器をリモート状態に設定します。

10. PciGpibExExecTrigger

【機能】

指定した機器に対してトリガ設定を行います。

【書式】

●C 言語

```
int PciGpibExExecTrigger (
    int      BoardNo,
    int*     AdrsTbl
);
```

●Visual Basic

```
Declare Function PciGpibExExecTrigger Lib "gpc4304.dll"( _
    ByVal    BoardNo    As Long,                  _
    ByRef    AdrsTbl    As Long                   _
) As Long
```

●Delphi

```
function PciGpibExExecTrigger (
        BoardNo      : Integer;
        AdrsTbl      : Pointer
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

AdrsTbl

機器のアドレステーブルを指定します。
デバイストリガ機能を動作させる機器のアドレステーブルへのポインタ（参照渡し）を指定します。アドレステーブルへの終端には必ず-1 を設定します。

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

・ 本関数はコントローラ・イン・チャージでのみ使用できます。

【使用例】

●C 言語

```
int Ret;
int AdrsTbl[2];

AdrsTbl[0] = 2;
AdrsTbl[1] = -1;
Ret = PciGpibExExecTrigger( 0, AdrsTbl );
```

●Visual Basic

```
Dim Ret As Long
Dim AdrsTbl(1) As Long

AdrsTbl(0) = 2
AdrsTbl(1) = -1
Ret = PciGpibExExecTrigger( 0, AdrsTbl(0) )
```

●Delphi

```
var
  Ret : Integer;
  AdrsTbl : Array[0..4] of Integer;

AdrsTbl[0] := 2;
AdrsTbl[1] := -1;
Ret := PciGpibExExecTrigger( 0, @AdrsTbl[0] );
```

ボードアクセス番号 0 のインタフェースモジュールから AdrsTbl に記述した機器に対してトリガ設定を行います。

11. PciGpibExExecDevClear

【機能】

全ての機器に対してデバイスクリア機能(装置の初期化)を動作させせます。

【書式】

●C 言語

```
int PciGpibExExecDevClear (
    int       BoardNo
);
```

●Visual Basic

```
Declare Function PciGpibExExecDevClear Lib "gpc4304.dll"( _
    ByVal   BoardNo    As Long    _
) As Long
```

●Delphi

```
function PciGpibExExecDevClear (
        BoardNo   : Integer
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

・　本関数はコントローラ・イン・チャージでのみ使用できます。

【使用例】

●C 言語

```
int Ret;
Ret = PciGpibExExecDevClear( 0 );
```

●Visual Basic

```
Dim Ret As Long
Ret = PciGpibExExecDevClear( 0 )
```

●Delphi

```
var
  Ret : Integer;

Ret := PciGpibExExecDevClear( 0 );
```

ボードアクセス番号０のインタフェースモジュールからデバイスクリアの送出を行います。

12. PciGpibExExecSdc

【機能】

指定した機器に対してデバイスクリア機能(装置の初期化)を動作させせます。

【書式】

●C 言語

```
int PciGpibExExecSdc (
    int        BoardNo,
    int*       AdrsTbl
);
```

●Visual Basic

```
Declare Function PciGpibExExecSdc Lib "gpc4304.dll"( _
    ByVal    BoardNo    As Long,                _
    ByRef    AdrsTbl    As Long                 _
) As Long
```

●Delphi

```
function PciGpibExExecSdc (
        BoardNo     : Integer;
        AdrsTbl     : Pointer
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

AdrsTbl

機器のアドレステーブルを指定します。
デバイスクリア機能を動作させる機器のアドレステーブルへのポインタ（参照渡し）を指定します。アドレステーブルへの終端には必ず-1 を設定します。

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

・ 本関数はコントローラ・イン・チャージでのみ使用できます。

【使用例】

●C 言語

```
int Ret;
int AdrsTbl[2];

AdrsTbl[0] = 2;
AdrsTbl[1] = -1;
Ret = PciGpibExExecSdc( 0, AdrsTbl );
```

●Visual Basic

```
Dim Ret As Long
Dim AdrsTbl(1) As Long

AdrsTbl(0) = 2
AdrsTbl(1) = -1
Ret = PciGpibExExecSdc( 0, AdrsTbl(0) )
```

●Delphi

```
var
  Ret : Integer;
  AdrsTbl : Array[0..4] of Integer;

AdrsTbl[0] := 2;
AdrsTbl[1] := -1;
Ret := PciGpibExExecSdc( 0, @AdrsTbl[0] );
```

ボードアクセス番号 0 のインタフェースモジュールから指定した機器に対してセレクテッドデバイスクリアの送出を行います。

13. PciGpibExSetLocal

【機能】

指定した機器をローカル状態にします（フロント・パネル制御可)。

【書式】

●C 言語

```
int PciGpibExSetLocal (
    int         BoardNo,
    int*        AdrsTbl
);
```

●Visual Basic

```
Declare Function PciGpibExSetLocal Lib "gpc4304.dll"( _
    ByVal    BoardNo    As Long,                    _
    ByRef    AdrsTbl    As Long                     _
) As Long
```

●Delphi

```
function PciGpibExSetLocal (
        BoardNo   : Integer;
        AdrsTbl   : Pointer
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

AdrsTbl

機器のアドレステーブルを指定します。
ローカル状態にさせる機器のアドレステーブルへのポインタ(参照渡し)を指定します。
アドレステーブルへの終端には必ず-1 を設定します。

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

・　本関数はコントローラ・イン・チャージでのみ使用できます。

【使用例】

●C 言語

```
int Ret;
int AdrsTbl[2];

AdrsTbl[0] = 2;
AdrsTbl[1] = -1;
Ret = PciGpibExSetLocal( 0, AdrsTbl );
```

●Visual Basic

```
Dim Ret As Long
Dim AdrsTbl(1) As Long

AdrsTbl(0) = 2
AdrsTbl(1) = -1
Ret = PciGpibExSetLocal( 0, AdrsTbl(0) )
```

●Delphi

```
var
  Ret : Integer;
  AdrsTbl : Array[0..4] of Integer;

AdrsTbl[0] := 2;
AdrsTbl[1] := -1;
Ret := PciGpibExSetLocal( 0, @AdrsTbl[0] );
```

ボードアクセス番号０のインタフェースモジュールから指定した機器に対してローカル状態の設定を行います。

14. PciGpibExSetLlo

【機能】

全ての機器をローカル・ロックアウト状態(フロントパネルからの操作を一切禁止の状態)にします。

【書式】

●C 言語

```
int PciGpibExSetLlo (
    int         BoardNo
);
```

●Visual Basic

```
Declare Function PciGpibExSetLlo Lib "gpc4304.dll"( _
    ByVal    BoardNo   As Long    _
) As Long
```

●Delphi

```
function PciGpibExSetLlo (
        BoardNo      : Integer
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

- 本関数はコントローラ・イン・チャージでのみ使用できます。
- 機器によっては、ローカルロックアウト状態を解除するためには、電源を再投入しなければならないものもあります。

【使用例】

●C 言語

```
int Ret;
Ret = PciGpibExSetLlo( 0 );
```

●Visual Basic

```
Dim Ret As Long
Ret = PciGpibExSetLlo( 0 )
```

●Delphi

```
var
  Ret : Integer;

Ret := PciGpibExSetLlo( 0 );
```

ローカル・ロックアウトの送出を行います。

15.　PciGpibExSetRwls

【機能】

指定した機器をリモート・ロックアウト状態にします。

【書式】

●C 言語

```
int PciGpibExSetRwls (
    int      BoardNo,
    int*     AdrsTbl
);
```

●Visual Basic

```
Declare Function PciGpibExSetRwls Lib "gpc4304.dll"( _
    ByVal     BoardNo    As Long,              _
    ByRef     AdrsTbl    As Long               _
) As Long
```

●Delphi

```
function PciGpibExSetRwls (
        BoardNo     : Integer;
        AdrsTbl     : Pointer
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

AdrsTbl

機器のアドレステーブルを指定します。
リモート･ロックアウト状態にさせる機器のアドレステーブルへのポインタ(参照渡し)を指定します。アドレステーブルへの終端には必ず-1 を設定します。

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

・　本関数はシステムコントローラでのみ使用できます。

【使用例】

●C 言語

```
int Ret;
int AdrsTbl[2];

AdrsTbl[0] = 2;
AdrsTbl[1] = -1;
Ret = PciGpibExSetRwls( 0, AdrsTbl );
```

●Visual Basic

```
Dim Ret As Long
Dim AdrsTbl(1) As Long

AdrsTbl(0) = 2
AdrsTbl(1) = -1
Ret = PciGpibExSetRwls( 0, AdrsTbl(0) )
```

●Delphi

```
var
  Ret : Integer;
  AdrsTbl : Array[0..4] of Integer;

AdrsTbl[0] := 2;
AdrsTbl[1] := -1;
Ret := PciGpibExSetRwls( 0, @AdrsTbl [0] );
```

ボードアクセス番号０のインタフェースモジュールから指定した機器に対してリモート・ロックアウトの設定を行います。

16.　PciGpibExExecPassCtrl

【機能】

指定した機器に対してパス・コントロールを行います。

【書式】

●C 言語

```
int PciGpibExExecPassCtrl (
    int      BoardNo,
    int*     AdrsTbl
);
```

●Visual Basic

```
Declare Function PciGpibExExecPassCtrl Lib "gpc4304.dll"( _
    ByVal    BoardNo    As Long,                _
    ByRef    AdrsTbl    As Long                 _
) As Long
```

●Delphi

```
function PciGpibExExecPassCtrl (
        BoardNo   : Integer;
        AdrsTbl   : Pointer
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

AdrsTbl

機器のアドレステーブルを指定します。
コントロールを渡す機器のアドレステーブルへのポインタ（参照渡し）を指定します。
アドレステーブルへの終端には必ず-1 を設定します。

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

- 本関数はコントローラ・イン・チャージでのみ使用できます。
- 本関数を実行することにより、相手にコントローラ機能があるかどうかに関係なく、本インタフェースモジュールはコントローラ・イン・チャージが解除されます。

【使用例】

●C 言語

```
int Ret;
int AdrsTbl[2];

AdrsTbl[0] = 2;
AdrsTbl[1] = -1;
Ret = PciGpibExExecPassCtrl( 0, AdrsTbl );
```

●Visual Basic

```
Dim Ret As Long
Dim AdrsTbl(1) As Long

AdrsTbl(0) = 2
AdrsTbl(1) = -1
Ret = PciGpibExExecPassCtrl( 0, AdrsTbl(0) )
```

●Delphi

```
var
  Ret : Integer;
  AdrsTbl : Array[0..4] of Integer;

AdrsTbl[0] := 2;
AdrsTbl[1] := -1;
Ret := PciGpibExExecPassCtrl( 0, @AdrsTbl[0] );
```

ボードアクセス番号 0 のインタフェースモジュールから指定した機器に対してパス・コントロールを実行します。

17. PciGpibExExecFindListener

【機能】

バス上に接続されているリスナ(デバイス)を探します。(IEEE 488.2 関数)

【書式】

●C 言語

```
int PciGpibExExecFindListener (
    int     BoardNo,
    int*    AdrsTbl,
    int*    FindAdrsTbl,
    int*    FindCnt
);
```

●Visual Basic

```
Declare Function PciGpibExExecFindListener Lib "gpc4304.dll"( _
    ByVal   BoardNo       As Long,    _
    ByRef   AdrsTbl       As Long,    _
    ByRef   FindAdrsTbl   As Long,    _
    ByRef   FindCnt       As Long     _
) As Long
```

●Delphi

```
function PciGpibExExecFindListener (
        BoardNo      : Integer;
        AdrsTbl      : Pointer;
        FindAdrsTbl  : Pointer;
        FindCnt      : Pointer
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

AdrsTbl

接続されているか確認したいリスナ(デバイス)の GP-IB アドレスを格納したテーブルへのポインタ（参照渡し）を指定します。
アドレステーブルへの終端には必ず-1 を設定します。

FindAdrsTbl

見つかった機器のアドレスを格納するテーブルへのポインタ(参照渡し)を指定します。
本 API 実行後、AdrsTbl で指定した機器が見つかった場合、その機器のアドレスが格納されます。
この引数に指定するテーブルのサイズは、AdrsTbl テーブル以上の格納領域を確保しておく必要があります。
本 API 呼び出し後、自動で終端に-1 が付加されます。

FindCnt

見つかった機器の総数を格納する変数へのポインタ（参照渡し）を指定します。

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

・　本関数はコントローラ・イン・チャージでのみ使用できます。

【使用例】

●C 言語

```
int Ret;
int AdrsTbl[2];
int FindAdrsTbl[2];
int FindCnt

AdrsTbl[0] = 2;
AdrsTbl[1] = -1;
Ret = PciGpibExExecFindListener( 0, AdrsTbl, FindAdrsTbl, &FindCnt );
```

●Visual Basic

```
Dim Ret As Long
Dim AdrsTbl(1) As Long
Dim FindAdrsTbl(1) As Long
Dim FindCnt As Long

AdrsTbl(0) = 2
AdrsTbl(1) = -1
Ret = PciGpibExExecFindListener( 0, AdrsTbl(0), FindAdrsTbl(0), FindCnt )
```

●Delphi

```
var
  Ret : Integer;
  AdrsTbl : Array[0..4] of Integer;
  FindAdrsTbl : Array[0..4] of Integer;
  FindCnt : Integer;

AdrsTbl[0] := 2;
AdrsTbl[1] := -1;
Ret := PciGpibExExecFindListener( 0, @AdrsTbl[0], @FindAdrsTbl[0], @FindCnt );
```

ボードアクセス番号 0 のインタフェースモジュールから'AdrsTbl'に格納されているアドレスの探索を実行し、見つかった機器アドレスとその総数を取得します。

18.　PciGpibExExecDevReset

【機能】

指定した機器を完全にリセットします。（IEEE 488.2 関数）

【書式】

●C 言語

```
int PciGpibExExecDevReset (
    uint       BoardNo,
    int*       AdrsTbl
);
```

●Visual Basic

```
Declare Function PciGpibExExecDevReset Lib "gpc4304.dll"( _
    ByVal   BoardNo  As Long,    _
    ByRef   AdrsTbl  As Long     _
) As Long
```

●Delphi

```
function PciGpibExExecDevReset (
        BoardNo    : Integer;
        AdrsTbl    : Pointer
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

AdrsTbl

機器のアドレステーブルを指定します。
リセット機能を動作させる機器のアドレステーブルへのポインタ（参照渡し）を指定します。アドレステーブルへの終端には必ず-1 を設定します。

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

・ 本関数はシステムコントローラでのみ使用できます。
・ 本関数は次の処理を行います。

1. バスの初期化
   IFC、次に REN がアクティブになります。その結果、全てのデバイスはアドレスされていない状態になり、GP-IB インタフェースモジュール（システムコントローラ）がコントローラ・イン・チャージ(CIC)になります。
2. メッセージ交換初期化

DCL バスコマンドが全ての接続されたデバイスに送られます。その結果として全ての IEEE 488.2 規格にあったデバイスは次の Reset(RST)メッセージを受信できるようになります。

3. デバイス初期化
   *RST メッセージが引数'AdrsTbl'にて指定されるアドレスのデバイスに送られます。その結果として、デバイス特有の機能が初期化されます。

【使用例】

●C 言語

```
int Ret;
int AdrsTbl[2];

AdrsTbl[0] = 2;
AdrsTbl[1] = -1;
Ret = PciGpibExExecDevReset( 0, AdrsTbl );
```

●Visual Basic

```
Dim Ret As Long
Dim AdrsTbl(1) As Long

AdrsTbl(0) = 2
AdrsTbl(1) = -1
Ret = PciGpibExExecDevReset( 0, AdrsTbl(0) )
```

●Delphi

```
var
  Ret : Integer;
  AdrsTbl : Array[0..4] of Integer;

AdrsTbl[0] := 2;
AdrsTbl[1] := -1;
Ret := PciGpibExExecDevReset( 0, @AdrsTbl [0] );
```

ボードアクセス番号 0 のインタフェースモジュールから指定した機器に対してリセットの実行を行います。

19. PciGpibExReSysCtrl

【機能】

システムコントローラの要求または解除を行います。

【書式】

●C 言語

```
int PciGpibExReSysCtrl (
    int     BoardNo,
    int     Mode
);
```

●Visual Basic

```
Declare Function PciGpibExReSysCtrl Lib "gpc4304.dll"( _
    ByVal    BoardNo    As Long,    _
    ByVal    Mode       As Long     _
) As Long
```

●Delphi

```
function PciGpibExReSysCtrl (
        BoardNo   : Integer;
        Mode      : Integer
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

Mode

システムコントローラの要求または解除のモードを指定します。

パラメータ	説明
0	システムコントローラの解除を行います。
1	システムコントローラの要求を行います。

【戻り値】

正常終了した場合は、0 が返されます。

0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

・ 本関数はシステムコントローラでのみ使用できます。

【使用例】

●C 言語

```
int Ret;
Ret = PciGpibExReSysCtrl( 0, 1 );
```

●Visual Basic

```
Dim Ret As Long
Ret = PciGpibExReSysCtrl( 0, 1 )
```

●Delphi

```
var
  Ret : Integer;

Ret := PciGpibExReSysCtrl( 0, 1 );
```

ボードアクセス番号０のインタフェースモジュールからシステムコントローラの要求を行います。

20. PciGpibExGoStandby

【機能】

インタフェースモジュールの動作状態をコマンドモードからデータモードへ遷移させます。

【書式】

●C 言語

```
int PciGpibExGoStandby (
    int         BoardNo,
    int         Mode
);
```

●Visual Basic

```
Declare Function PciGpibExGoStandby Lib "gpc4304.dll"( _
    ByVal     BoardNo          As Long,          _
    ByVal     Mode             As Long           _
) As Long
```

●Delphi

```
function PciGpibExGoStandby (
        BoardNo     : Integer;
        Mode        : Integer
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

Mode

データモードを指定します。

パラメータ	説明
0	通常のデータモードに遷移します。
1	シャドウ・ハンドシェークモードに遷移します。 本モード指定時、機器間のデータ転送を監視し、デリミタ検出時には自動でコマンドモードに遷移します。デリミタ指定には、受信デリミタの設定が使用されます。

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

・ 本関数はコントローラ・イン・チャージでのみ使用できます。
・ 本関数でシャドウ・ハンドシェークモードを使用することにより、自分自身が関与しない機器間でのデータ転送を行うことができます。また機器間でのデータ転送において、デリミタ検出時に自動でコマンドモードに戻ることができます。

・ 本関数を使用してシャドウ・ハンドシェークを行う場合、デリミタ指定は必ず行うようにしてください。

【使用例】

●C 言語

```
int Ret;
Ret = PciGpibExGoStandby( 0, 0 );
```

●Visual Basic

```
Dim Ret As Long
Ret = PciGpibExGoStandby( 0, 0 )
```

●Delphi

```
var
  Ret : Integer;

Ret := PciGpibExGoStandby( 0, 0 );
```

ボードアクセス番号0のインタフェースモジュールの動作モードを通常のデータモードに遷移させます。

21. PciGpibExGoActCtrller

【機能】

インタフェースモジュールの動作状態をデータモードからコマンドモードへ遷移させせます。

【書式】

●C 言語

```
int PciGpibExGoActCtrller (
    int         BoardNo,
    int         Mode
);
```

●Visual Basic

```
Declare Function PciGpibExGoActCtrller Lib "gpc4304.dll"( _
    ByVal    BoardNo        As Long,        _
    ByVal    Mode           As Long         _
) As Long
```

●Delphi

```
function PciGpibExGoActCtrller (
        BoardNo    : Integer;
        Mode       : Integer
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

Mode

コマンドモードへの遷移方法を指定します。

パラメータ	説明
0	非同期にてコマンドモードへ遷移します。
1	同期してコマンドモードへ遷移します。

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】
- 本関数はコントローラ・イン・チャージでのみ使用できます。
- 本関数は PciGpibExGoStandby 関数で通常のデータモードに遷移後にアプリケーションプログラムでコマンドモードにするために使用します。
- Mode=1(同期してコマンドモードへ遷移)を使用する場合には､インタフェースモジュールはリスナ状態である必要があります。
- インタフェースモジュールがトーカ状態もしくはデータ転送エラー発生後などの場合には、Mode=0(非同期にてコマンドモードへ遷移)を使用してください。
  その場合、データ転送が完了しているかどうかに関係無く、ただちにコマンドモードに遷移します。そのため、データ転送完了を確認後に実行するようにしてください。
- データ転送実行中に､Mode=0(非同期にてコマンドモードへ遷移)で本 API を実行した場合、タイミングによっては転送中のデータが失われる可能性があります。

【使用例】

●C 言語

```
int Ret;
Ret = PciGpibExGoActCtrller( 0, 0 );
```

●Visual Basic

```
Dim Ret As Long
Ret = PciGpibExGoActCtrller( 0, 0 )
```

●Delphi

```
var
  Ret : Integer;

Ret := PciGpibExGoActCtrller( 0, 0 );
```

動作モードをデータモードからコマンドモードへ非同期にて遷移させます。

22. PciGpibExExecSpoll

【機能】

指定した機器に対してシリアル・ポーリングを行い、ステータス・バイトの受信を行います。

【書式】

●C 言語

```
int PciGpibExExecSpoll (
    int     BoardNo,
    int*    AdrsTbl,
    int*    StbTbl,
    int*    StbAdrs
);
```

●Visual Basic

```
Declare Function PciGpibExExecSpoll Lib "gpc4304.dll"( _
    ByVal   BoardNo   As Long,   _
    ByRef   AdrsTbl   As Long,   _
    ByRef   StbTbl    As Long,   _
    ByRef   StbAdrs   As Long    _
) As Long
```

●Delphi

```
function PciGpibExExecSpoll (
        BoardNo    : Integer;
        AdrsTbl    : Pointer;
        StbTbl     : Pointer;
        StbAdrs    : Pointer
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

AdrsTbl

シリアル・ポールを行う機器のアドレステーブルへのポインタ（参照渡し）を指定します。

アドレステーブルへの終端には必ず-1 を設定します。

StbTbl

シリアル・ポーリングを行い受信した各機器の有効なステータス・バイトを格納するテーブルへのポインタ（参照渡し）を指定します。

本関数呼び出し後、終端に-1 が付加されます。

StbAdrs

SRQ を送出し、有効なステータス・バイトを持っていた各機器のアドレスを格納するテーブルへのポインタ（参照渡し）を指定します。

本関数呼び出し後、終端に-1 が付加されます。

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

- 本関数はコントローラ・イン・チャージでのみ使用できます。
- 本関数を呼び出す際は、PciGpibExCheckSrq 関数もしくは PciGpibExWaitSignal 関数にて、機器よりの SRQ を受け付けていることを確認してから呼び出してください。
- アドレステーブルに複数の機器アドレスを指定することにより、複数の機器に対してシリアル・ポーリングを行うことが可能です。
  ステータス・バイト格納テーブルと SRQ 送出機器アドレス格納テーブルには、機器の数＋終端を格納することができる領域を確保しておく必要があります。

【使用例】

●C 言語

```
int Ret;
int AdrsTbl[2];
int StbTbl[2];
int StbAdrs[2];

AdrsTbl[0] = 2;
AdrsTbl[1] = -1;
Ret = PciGpibExExecSpoll( 0, AdrsTbl, StbTbl, StbAdrs );
```

●Visual Basic

```
Dim nRet As Long
Dim AdrsTbl(1) As Long
Dim StbTbl(1) As Long
Dim StbAdrs(1) As Long

AdrsTbl(0) = 2
AdrsTbl(1) = -1
Ret = PciGpibExExecSpoll( 0, AdrsTbl(0), StbTbl(0), StbAdrs(0) )
```

●Delphi

```
var
  Ret : Integer;
  AdrsTbl : Array[0..4] of Integer;
  StbTbl : Array[0..4] of Integer;
  StbAdrs : Array[0..4] of Integer;

AdrsTbl[0] := 2;
AdrsTbl[1] := -1;
Ret := PciGpibExExecSpoll( 0, @AdrsTbl[0], @StbTbl[0], @StbAdrs[0]);
```

ボードアクセス番号0のインタフェースモジュールから AdrsTbl に記述される各機器に対してシリアル・ポーリングを実行します。

23. PciGpibExCheckSrq

【機能】

SRQ 信号受信フラグの有効/無効を確認します。

【書式】

●C 言語

```
int PciGpibExCheckSrq (
   int       BoardNo
);
```

●Visual Basic

```
Declare Function PciGpibExCheckSrq Lib "gpc4304.dll"( _
   ByVal  BoardNo    As Long                       _
) As Long
```

●Delphi

```
function PciGpibExCheckSrq (
      BoardNo  : Integer
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

【戻り値】

正常終了しますと、以下の値を返します。

戻り値	意味
5	SRQ信号を受け付けていません。
4	SRQ信号を受け付けています。

上記以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

- 本関数はコントローラ・イン・チャージ/非コントローラ・イン・チャージの両方の状態にて使用できます。
- PciGpibExInitBoard 関数実行後は SRQ 受信割り込み許可状態となっています。
- 本関数を実行後、機器からの SRQ 信号がアサート(有効)になっている場合には、機器からのステータスバイトを受信するために、必ず PciGpibExExecSpoll 関数を実行してください。

【使用例】

●C 言語

```
int Ret;
Ret = PciGpibExCheckSrq( 0 );
```

●Visual Basic

```
Dim Ret As Long
Ret = PciGpibExCheckSrq( 0 )
```

●Delphi

```
var
  Ret : Integer;

Ret := PciGpibExCheckSrq( 0 );
```

SRQ 受信フラグの有効/無効を確認します。

24. PciGpibExClearSrq

【機能】

SRQ 受信フラグのクリアを行います。

【書式】

●C 言語

```
int PciGpibExClearSrq (
   int       BoardNo
);
```

●Visual Basic

```
Declare Function PciGpibExClearSrq Lib "gpc4304.dll"( _
   ByVal   BoardNo  As Long    _
) As Long
```

●Delphi

```
function PciGpibExClearSrq (
      BoardNo   : Integer
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

【戻り値】

正常終了した場合は、0 が返されます。

0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

- 本関数はコントローラ・イン・チャージでのみ使用できます。
- SRQ 受信フラグが有効にも関わらず、シリアル・ポーリング実行後、SRQ 発信機器が検出できなかった場合において、本関数にて強制的に SRQ 受信フラグのクリアを行います。
- 本関数にてソフトウェアが保持している SRQ 受信フラグのクリアを行いますが、これによって、GP-IB バスラインの SRQ 信号がデアサート(信号無効状態)にはなりません。
- 意図しない GP-IB バスラインの SRQ 信号アサート状態を解除するためには、SRQ 信号をアサートした該当機器が SRQ 信号をデアサートさせるか、または全ての機器の GP-IB インタフェース機能を再初期化させる必要があります。

【使用例】

●C 言語

```
int Ret;
Ret = PciGpibExClearSrq(0);
```

●Visual Basic

```
Dim Ret As Long
Ret = PciGpibExClearSrq(0)
```

●Delphi

```
var
  Ret : Integer;

Ret := PciGpibExClearSrq(0);
```

SRQ 受信フラグをクリアします。

25. PciGpibExEnableSrq

【機能】

SRQ 受信割り込みを許可します。

【書式】

●C 言語

```
int PciGpibExEnableSrq (
    int         BoardNo
);
```

●Visual Basic

```
Declare Function PciGpibExEnableSrq Lib "gpc4304.dll"( _
    ByVal   BoardNo   As Long                 _
) As Long
```

●Delphi

```
function PciGpibExEnableSrq (
        BoardNo    : Integer;
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

- 本関数はコントローラ・イン・チャージでのみ使用できます。
- 本関数呼び出し後に SRQ 信号がアサートされた時、PciGpibExCheckSrq 関数にて SRQ 受信フラグが有効となります。
- PciGpibExInitBoard 関数実行後は SRQ 受信割り込み許可状態となっています。

【使用例】

●C 言語

```
int Ret;
Ret = PciGpibExEnableSrq( 0 );
```

●Visual Basic

```
Dim Ret As Long
Ret = PciGpibExEnableSrq( 0 )
```

●Delphi

```
var
  Ret : Integer;

Ret := PciGpibExEnableSrq( 0 );
```

SRQ 受信割り込みを許可します。

26. PciGpibExDisableSrq

【機能】

SRQ 受信割り込みを禁止します。

【書式】

●C 言語

```
int PciGpibExDisableSrq (
    int        BoardNo
);
```

●Visual Basic

```
Declare Function PciGpibExDisableSrq Lib "gpc4304.dll"( _
    ByVal   BoardNo   As Long                 _
) As Long
```

●Delphi

```
function PciGpibExDisableSrq (
        BoardNo   : Integer
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

・ 本関数はコントローラ・イン・チャージでのみ使用できます。
・ 本関数実行後は、SRQ 受信割込みは禁止となるため、PciGpibExCheckSrq 関数による SRQ 受信の確認はできません。
・ 本関数実行後に SRQ 信号がアサートされ、SRQ 信号アサート状態のままで PciGpibExEnableSrq 関数を呼び出しますと、SRQ 受信割込みが有効となるため、PciGpibExCheckSrq 関数にて SRQ 受信が有効となります。

【使用例】

●C 言語

```
int Ret;
Ret = PciGpibExDisableSrq( 0 );
```

●Visual Basic

```
Dim Ret As Long
Ret = PciGpibExDisableSrq( 0 )
```

●Delphi

```
var
  Ret : Integer;

Ret := PciGpibExDisableSrq( 0 );
```

SRQ 受信割り込みを禁止します。

27. PciGpibExExecPpoll

【機能】

パラレルポーリングを実行します。

【書式】

●C 言語

```
int PciGpibExExecPpoll (
   int        BoardNo,
   int*       Pst
);
```

●Visual Basic

```
Declare Function PciGpibExExecPpoll Lib "gpc4304.dll"( _
   ByVal    BoardNo    As Long,              _
   ByRef    Pst        As Long               _
) As Long
```

●Delphi

```
function PciGpibExExecPpoll (
            BoardNo         : Integer;
   var      Pst             : Integer
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

Pst

パラレルポーリング時の応答ステータスを格納する変数へのポインタ（参照渡し）を指定します。

各機器毎の情報を得るために、事前に各機器に対してどのビットを使用するのか、また応答の極性をどうするのかについては、PciGpibExCfgPpoll 関数で指定する方法(リモートコンフィグレーション)と機器のフロントパネルからの設定(ローカルコンフィグレーション)の 2 通りがあります。

【戻り値】

正常終了した場合は、0 が返されます。

0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

・ 本関数はコントローラ・イン・チャージでのみ使用できます。

【使用例】

●C 言語

```
int Ret;
int Pst;
Ret = PciGpibExExecPpoll( 0, &Pst );
```

●Visual Basic

```
Dim Ret As Long
Dim Pst As Long
Ret = PciGpibExExecPpoll( 0, Pst )
```

●Delphi

```
var
  Ret : Integer;
  Pst : Integer;

Ret := PciGpibExecPpoll( 0, Pst );
```

パラレル・ポーリングを実行します。

28. PciGpibExCfgPpoll

【機能】

機器に対してリモートにてパラレルポール・コンフィグレーションを行います。
コンフィグレーションは応答する DIO ラインの割り当てと、その極性を設定します。
(リモート・コンフィグレーション)

【書式】

●C 言語

```
int PciGpibExCfgPpoll (
    int         BoardNo,
    int*        AdrsTbl,
    int         Ppe
);
```

●Visual Basic

```
Declare Function PciGpibExCfgPpoll Lib "gpc4304.dll"( _
    ByVal    BoardNo    As Long,                _
    ByRef    AdrsTbl    As Long,                _
    ByVal    Ppe        As Long                 _
) As Long
```

●Delphi

```
function PciGpibExCfgPpoll (
        BoardNo     : Integer;
        AdrsTbl     : Pointer;
        Ppe         : Integer
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

AdrsTbl

パラレルポールに応答させる機器のアドレステーブルへのポインタ（参照渡し）を指定します。
アドレステーブルへの終端には必ず-1 を設定します。

Ppe

パラレルポール応答設定を指定します。

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
0	0	0	0	S	p3	p2	p1

p3	p2	p1	DIO ライン 1～8 の指定
0	0	0	DIO1 で応答
0	0	1	DIO2 で応答
0	1	0	DIO3 で応答
0	1	1	DIO4 で応答
1	0	0	DIO5 で応答
1	0	1	DIO6 で応答
1	1	0	DIO7 で応答
1	1	1	DIO8 で応答

S	極性の指定
0	機器の ist ビットがクリア(0)のとき、割り当てた応答 DIO ラインをアサートします。
1	機器の ist ビットがセット(1)のとき、割り当てた応答 DIO ラインをアサートします。

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

・　本関数はコントローラ・イン・チャージでのみ使用できます。

【使用例】

●C 言語

```
int Ret;
int AdrsTbl[2];

AdrsTbl[0] = 2;
AdrsTbl[1] = -1;
Ret = PciGpibExCfgPpoll( 0, AdrsTbl, 0x09 );
```

●Visual Basic

```
Dim Ret As Long
Dim AdrsTbl(1) As Long

AdrsTbl(0) = 2
AdrsTbl(1) = -1
Ret = PciGpibExCfgPpoll( 0, AdrsTbl(0), &H09 )
```

●Delphi

```
var
  Ret : Integer;
  AdrsTbl : Array[0..4] of Integer;

AdrsTbl[0] := 2;
AdrsTbl[1] := -1;
Ret := PciGpibExCfgPpoll( 0, @AdrsTbl[0], $09 );
```

ボードアクセス番号０のインタフェースモジュールから、指定した機器に対してリモートにてパラレルポール・コンフィグレーションを行います。

29. PciGpibExUnCfgPpoll

【機能】

リモート・コンフィグレーションにて、パラレルポール・コンフィグレーションされている機器に対して PPD(Parallel Poll Disable)を送出し、パラレルポール応答設定を解除します。

【書式】

●C 言語

```
int PciGpibExUnCfgPpoll (
    int         BoardNo,
    int*        AdrsTbl
);
```

●Visual Basic

```
Declare Function PciGpibExUnCfgPpoll Lib "gpc4304.dll"( _
    ByVal    BoardNo    As Long,              _
    ByRef    AdrsTbl    As Long               _
) As Long
```

●Delphi

```
function PciGpibExUnCfgPpoll (
        BoardNo       : Integer;
        AdrsTbl       : Pointer
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

AdrsTbl

パラレルポール応答設定を解除する機器のアドレステーブルへのポインタ（参照渡し）を指定します。
アドレステーブルへの終端には必ず-1 を設定します。
アドレステーブルの先頭に-1 を設定した場合は、PPU(Parallel Poll Unconfigure)を送出してリモート・コンフィギュレーション機能を持つ全ての機器の応答設定を解除します。

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】
- 本関数はコントローラ・イン・チャージでのみ使用できます。
- 本関数はリモート・コンフィグレーションにて、パラレルポール・コンフィグレーションされている機器に対して PPD(Parallel Poll Disable)を送出し、パラレルポール応答設定を解除します。

【使用例】

●C 言語

```
int Ret;
int AdrsTbl[2];

AdrsTbl[0] = 2;
AdrsTbl[1] = -1;
Ret = PciGpibExUnCfgPpoll( 0, AdrsTbl );
```

●Visual Basic

```
Dim Ret As Long
Dim AdrsTbl(1) As Long

AdrsTbl(0) = 2
AdrsTbl(1) = -1
Ret = PciGpibExUnCfgPpoll( 0, AdrsTbl(0) )
```

●Delphi

```
var
  Ret : Integer;
  AdrsTbl : Array[0..4] of Integer;

AdrsTbl[0] := 2;
AdrsTbl[1] := -1;
Ret := PciGpibExUnCfgPpoll( 0, @AdrsTbl[0] );
```

ボードアクセス番号 0 のインタフェースモジュールから、パラレルポール応答設定解除を行います。

30. PciGpibExWriteBusCmd

【機能】

機器に対してバス・コマンド(マルチライン・インタフェース・メッセージ)を発行します。

【書式】

●C 言語

```
int PciGpibExWriteBusCmd (
    int         BoardNo,
    int*        CmdTbl
);
```

●Visual Basic

```
Declare Function PciGpibExWriteBusCmd Lib "gpc4304.dll"( _
    ByVal    BoardNo    As Long,          _
    ByRef    CmdTbl     As Long           _
) As Long
```

●Delphi

```
function PciGpibExWriteBusCmd (
        BoardNo   : Integer;
        CmdTbl    : Pointer
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

CmdTbl

バス上に送出されるコマンドを格納してあるテーブルへのポインタ（参照渡し）を指定します。
また、コマンドテーブルの終端には必ず-1 を設定してください。

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

・ 本関数はコントローラ・イン・チャージでのみ使用できます。
・ コマンドテーブルに設定されるデータは、そのままの形で GP-IB バスに送出されます。

【使用例】

●C 言語

```
int Ret;
int CmdTbl[3];

CmdTbl[0] = 0x3F;
CmdTbl[1] = 0x22;
CmtTbl[2] = -1;
Ret = PciGpibExWriteBusCmd( 0, CmdTbl );
```

●Visual Basic

```
Dim nRet As Long
Dim CmdTbl(2) As Long

CmdTbl(0) = &H3F
CmdTbl(1) = &H22
CmdTbl(2) = -1
nRet = PciGpibExWriteBusCmd( 0, npCmdTbl(0) )
```

●Delphi

```
var
  Ret : Integer;
  CmdTbl : Array[0..9] of Integer;

CmdTbl[0] := $3F;
CmdTbl[1] := $22;
CmtTbl[2] := -1;
Ret := PciGpibExWriteBusCmd( 0, @CmdTbl[0] );
```

ボードアクセス番号 0 のインタフェースモジュールから npCmdTbl に記述されているバス・コマンドを送出します。

31. PciGpibExSetSignal

【機能】

各種バス・ステータスにおける事象変化の検出条件の設定を行います。

【書式】

●C 言語

```
int PciGpibExSetSignal (
    int         BoardNo,
    UINT        Signal,
    BOOL        Detect
);
```

●Visual Basic

```
Declare Function PciGpibExSetSignal Lib "gpc4304.dll"( _
    ByVal    BoardNo    As Long,                 _
    ByVal    Signal     As Long,                 _
    ByVal    Detect     As Long                  _
) As Long
```

●Delphi

```
function PciGpibExSetSignal (
        BoardNo     : Integer;
        Signal      : UINT;
        Detect      : Boolean
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

Signal

検出条件を以下のビット情報で指定します。

bit31	IFC 受信検出許可 (非コントローラ・イン・チャージのみ)
bit30	SRQ 受信検出許可 (コントローラ・イン・チャージのみ)
bit29	シリアルポール終了検出許可 (非コントローラ・イン・チャージのみ)
bit28	デバイストリガ受信検出許可 (非コントローラ・イン・チャージのみ)
bit27	デバイスクリア受信検出許可 (非コントローラ・イン・チャージのみ)
bit26	データ受信検出許可 (非コントローラ・イン・チャージのみ)
bit25	リスナ指定検出許可 (非コントローラ・イン・チャージのみ)

bit24	トーカ指定検出許可 （非コントローラ・イン・チャージのみ）
bit23	入出力完了検出許可 （コントローラ・イン・チャージ/非コントローラ・イン・チャージ）
bit22	リモート状態検出許可 （非コントローラ・イン・チャージのみ）
bit21	ロックアウト状態検出許可 （非コントローラ・イン・チャージのみ）
bit20	デリミタ検出許可 （コントローラ・イン・チャージのみ）
bit19	コントローラ・イン・チャージ(CIC)検出許可 （コントローラ・イン・チャージのみ）
bit18	ATN 信号アクティブ検出許可 （コントローラ・イン・チャージのみ）
bit1～bit17	予約。0 を指定してください。
bit0	タイムアウト有り、0:タイムアウト無し

※ 1:検出許可，0:検出無効

複数の条件設定(OR 指定)ができます。
ただし、コントローラ・イン・チャージのみと非コントローラ・イン・チャージのみの条件は一緒には設定できません。

Detect
検出フラグを指定します。

パラメータ	説明
0	検出を無効にします。指定された検出項目の状態はクリアされ、事象変化検出を行いません。
1	検出を有効にします。指定された検出項目の設定を有効とし、事象変化検出を行います。

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

・ 弊社デバイスはデータ受信後、RFD ホールドオフ状態となりますので、以降、データ受信検出ビットが有効とならない場合があります。
非コントローラ状態でデータを受信する場合など、リスナ指定検出後にデータを受信するようにして下さい。

【使用例】

●C 言語

```
int Ret;
Ret = PciGpibExSetSignal( 0, 0x02000001, 1 );
```

●Visual Basic

```
Dim Ret As Long
Ret = PciGpibExSetSignal( 0, &h02000001, 1 )
```

●Delphi

```
var
  Ret : Integer;

Ret := PciGpibExSetSignal( 0, $02000001, 1 );
```

ボードアクセス番号０のインタフェースモジュールに対して、リスナ指定検出を許可、タイムアウト有りとして設定します。

32. PciGpibExWaitSignal

【機能】

各種バス・ステータスの事象変化を待ちます。

【書式】

●C 言語

```
int PciGpibExWaitSignal (
    int         BoardNo,
    UINT*       Signal
);
```

●Visual Basic

```
Declare Function PciGpibExWaitSignal Lib "gpc4304.dll"( _
    ByVal     BoardNo    As Long,            _
    ByRef     Signal     As Long             _
) As Long
```

●Delphi

```
function PciGpibExWaitSignal (
              BoardNo          : Integer;
    var       Signal           : Cardinal
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

Signal

検出された事象を格納する変数へのポインタ（参照渡し）を指定します。
バス・ステータスは以下のビット情報で格納されます。

bit31	IFC 受信検出 (非コントローラ・イン・チャージのみ)
bit30	SRQ 受信検出 (コントローラ・イン・チャージのみ)
bit29	シリアルポール終了検出 (非コントローラ・イン・チャージのみ)
bit28	デバイストリガ受信検出 (非コントローラ・イン・チャージのみ)
bit27	デバイスクリア受信検出 (非コントローラ・イン・チャージのみ)
bit26	データ受信検出 (非コントローラ・イン・チャージのみ)
bit25	リスナ指定検出許可 (非コントローラ・イン・チャージのみ)
bit24	トーカ指定検出 (非コントローラ・イン・チャージのみ)

bit23	入出力完了検出 （コントローラ・イン・チャージ/非コントローラ・イン・チャージ）
bit22	リモート状態検出 （非コントローラ・イン・チャージのみ）
bit21	ロックアウト状態検出 （非コントローラ・イン・チャージのみ）
bit20	デリミタ検出 （コントローラ・イン・チャージのみ）
bit19	コントローラ・イン・チャージ(CIC)検出 （コントローラ・イン・チャージのみ）
bit18	ATN 信号アクティブ検出 （コントローラ・イン・チャージのみ）
bit0～bit17	予約。0 が格納されます。

※ 1:検出, 0:未検出

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

・ 各モードで検出できない事象のビットが有効になることがあります。検出できない事象は無視するようにしてください。
・ 弊社デバイスはデータ受信後、RFD ホールドオフ状態となりますので、以降、データ受信検出ビットが有効とならない場合があります。
非コントローラ状態でデータを受信する場合など、リスナ指定検出後にデータを受信するようにして下さい。

【使用例】

●C 言語

```
int Ret;
UINT Status;
Ret = PciGpibExWaitSignal(0, &Status);
```

●Visual Basic

```
Dim Ret As Long
Dim Status As Long
Ret = PciGpibExWaitSignal(0, Status)
```

●Delphi

```
var
  Ret : Integer;
  Status : Cardinal;

Ret := PciGpibExWaitSignal(0, Status);
```

ボードアクセス番号 0 のインタフェースモジュールの事象変化検出を待ちます。

33. PciGpibExGetStatus

【機能】

現在のバス・ステータスを取得します。

【書式】

●C 言語

```
int PciGpibExGetStatus (
    int         BoardNo,
    UINT*       Status
);
```

●Visual Basic

```
Declare Function PciGpibExGetStatus Lib "gpc4304.dll"( _
    ByVal    BoardNo    As Long,                    _
    ByRef    Status     As Long                     _
) As Long
```

●Delphi

```
function PciGpibExGetStatus (
              BoardNo          : Integer;
    var       Status           : Cardinal
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

Signal

バス・ステータスを格納する変数へのポインタ（参照渡し）を指定します。
バス・ステータスは以下のビット情報で格納されます。

bit31	IFC 受信検出 （非コントローラ・イン・チャージのみ）
bit30	SRQ 受信検出 （コントローラ・イン・チャージのみ）
bit29	シリアルポール終了検出 （非コントローラ・イン・チャージのみ）
bit28	デバイストリガ受信検出 （非コントローラ・イン・チャージのみ）
bit27	デバイスクリア受信検出 （非コントローラ・イン・チャージのみ）
bit26	データ受信検 （非コントローラ・イン・チャージのみ）
bit25	リスナ指定検出 （非コントローラ・イン・チャージのみ）
bit24	トーカ指定検出 （非コントローラ・イン・チャージのみ）

bit23	入出力完了検出 （コントローラ・イン・チャージ/非コントローラ・イン・チャージ）
bit22	リモート状態検出 （非コントローラ・イン・チャージのみ）
bit21	ロックアウト状態検出 （非コントローラ・イン・チャージのみ）
bit20	デリミタ検出 （コントローラ・イン・チャージのみ）
bit19	コントローラ・イン・チャージ(CIC)検出 （コントローラ・イン・チャージのみ）
bit18	ATN 信号アクティブ検出 （コントローラ・イン・チャージのみ）
bit17	非同期入出力中に送信エラーを検出 （コントローラ・イン・チャージ/非コントローラ・イン・チャージ）
bit16	予約。0 が格納されます。
bit15	シリアルポールモードステータス[SPMS]検出 （コントローラ・イン・チャージ）
bit0～bit14	予約。0 が格納されます。

※ 1:検出, 0:未検出

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

- 各モードで検出できない事象のビットが有効になることがあります。検出できない事象は無視するようにしてください。
- 弊社デバイスはデータ受信後、RFD ホールドオフ状態となりますので、以降、データ受信検出ビットが有効とならない場合があります。
  非コントローラ状態でデータを受信する場合など、リスナ指定検出後にデータを受信するようにして下さい。
- bit24(トーカ指定)のクリアは、トーカ指定が解除された時に自動的に行われます。
- bit25(リスナ指定)のクリアは、リスナ指定が解除された時に自動的に行われます。
- bit26(データ受信)のクリアは、トーカ指定時、およびリスナ指定解除時に自動的に行われます。
- bit22(リモート状態)のクリアは、ローカル状態になったときに自動クリアされます。
- bit21(ロックアウト状態)のクリアは、ロックアウト解除状態時に自動クリアされます。
- bit30(SRQ 受信)のクリアは、シリアル・ポーリングを行うと自動クリアされます。
- bit15(SPMS)のクリアは、シリアルポールモードステータス解除時に自動クリアされます。

【使用例】

●C 言語

```
int Ret;
UINT Status;
Ret = PciGpibExGetStatus( 0, &Status );
```

●Visual Basic

```
Dim Ret As Long
Dim Status As Long
Ret = PciGpibExGetStatus( 0, Status )
```

●Delphi

```
var
  Ret : Integer;
  Status : Cardinal;

Ret := PciGpibExGetStatus( 0, Status );
```

バス・ステータスを取得します。

34. PciGpibExClrStatus

【機能】

バス・ステータス情報のクリアを行います。

【書式】

●C 言語

```
int PciGpibExClrStatus (
    int         BoardNo,
    UINT        Status
);
```

●Visual Basic

```
Declare Function PciGpibExClrStatus Lib "gpc4304.dll"( _
    ByVal    BoardNo    As Long,         _
    ByVal    Status     As Long          _
) As Long
```

●Delphi

```
function PciGpibExClrStatus (
        BoardNo     : Integer;
        Status      : UINT
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

Status

クリアするバス・ステータスを指定します。
バス・ステータスは以下のビット情報で指定します。

bit31	IFC 受信検出クリア
bit30	SRQ 受信検出クリア
bit29	シリアルポール終了検出クリア
bit28	デバイストリガ受信検出クリア
bit27	デバイスクリア受信検出クリア
bit26	データ受信検出クリア
bit25	リスナ指定検出クリア
bit24	トーカ指定検出クリア
bit23	入出力完了検出クリア
bit22	リモート状態検出クリア
bit21	ロックアウト状態検出クリア
bit20	デリミタ検出クリア
bit19	コントローラ・イン・チャージ(CIC)検出クリア
bit18	ATN 信号アクティブ検出クリア
bit17	非同期入出力中に送信エラーを検出クリア
bit0～bit16	予約。0 を指定してください。

※ 1:クリアします, 0:クリアしません

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

・ データ非同期転送進行中にbit23(入出力完了検出クリア)を有効として本API を呼び出した時、非同期転送は中断されず、保持されている入出力完了検出ステータスのみがクリアされます。進行中のデータ非同期転送を中断することはできません。
実際のデータ転送が行われていない場合、データ非同期転送はタイムアウトによって終了します。

【使用例】

●C 言語

```
int Ret;
Ret = PciGpibExClrStatus(0, 0x02000000);
```

●Visual Basic

```
Dim Ret As Long
Ret = PciGpibExClrStatus(0, &h02000000)
```

●Delphi

```
var
  Ret : Integer;

Ret := PciGpibExClrStatus(0, $02000000);
```

リスナ指定をクリアします。

35. PciGpibExMastSendData

【機能】

指定した機器に対して、データを送信します。

【書式】

●C 言語

```
int PciGpibExMastSendData (
  int      BoardNo,
  int*     AdrsTbl,
  DWORD    Length,
  void*    Buffer,
  UINT     Msg
);
```

●Visual Basic

バイト型/数値型変数のデータ送信の場合

```
Declare Function PciGpibExMastSendData Lib "gpc4304.dll"( _
  ByVal  BoardNo  As Long,                    _
  ByRef  AdrsTbl  As Long,                    _
  ByVal  Length   As Long,                    _
  ByRef  Buffer   As Any,                     _
  ByVal  Msg      As Long                     _
) As Long
```

文字列型変数のデータ送信の場合

```
Declare Function PciGpibExMastSendData Lib "gpc4304.dll"( _
  ByVal  BoardNo  As Long,     _
  ByRef  AdrsTbl  As Long,     _
  ByVal  Length   As Long,     _
  ByVal  Buffer   As String,   _
  ByVal  Msg      As Long      _
) As Long
```

非同期データ送信の場合

```
Declare Function PciGpibExMastSendData Lib "gpc4304.dll"( _
  ByVal  BoardNo  As Long,                   _
  ByRef  AdrsTbl  As Long,                   _
  ByVal  Length   As Long,                   _
  ByVal  Buffer   As Any,                    _
  ByVal  Msg      As Long                    _
) As Long
```

●Delphi

```
function PciGpibExMastSendData (
     BoardNo    : Integer;
     AdrsTbl    : Pointer;
     Length     : Cardinal;
     Buffer     : Pointer;
     Msg        : Cardinal
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

AdrsTbl

機器のアドレステーブルへのポインタ（参照渡し）を指定します。
アドレステーブルへの終端には必ず-1 を設定します。

Length

送信データ長のバイトサイズを指定します。

Buffer

送信バッファ領域へのポインタ（参照渡し）を指定します。

※ Visual Basic の場合には、API の Declare Function の宣言がバイト型/数値型と文字列型とでは異なります。

※ Visual Basic で非同期送信を行う場合は、GlobalAlloc 関数などを用いて割り当てたメモリ領域を指定してください。詳しくはサンプルプログラムをご参照ください。

Msg

非同期送信完了時にアプリケーションに通知されるメッセージを指定します。
同期動作時には WM_NULL もしくは 0 を指定してください

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

・ 本関数はコントローラ・イン・チャージでのみ使用できます。
・ 同期入出力時には、タイムアウト発生もしくは何らかの要因により処理が終了した場合において、本関数の処理を終了します。
・ 非同期入出力時は、本関数はデータ送信開始のみ行い、データ送信完了を待たずに本関数の処理を終了します。データ送信完了の確認は、PciGpibExGetStatus 関数において入出力完了を監視、または、アプリケーションに通知されるメッセージにより行います。その時点で、本関数を再度呼び出すことにより、送信ステータスの取得を行います。

- 本関数を非同期入出力完了後に呼び出した場合、送信ステータスを取得して処理を終了します。
  送信ステータスを取得後は、再度本関数により送信処理が行われるようになります。
- 機器からの応答がタイムアウト時間を経過しても無い場合には、タイムアウトエラーとして転送中断が行われます。
- 送信データ長として 0 を指定すると、マルチラインメッセージとして本インタフェースモジュールをトーカ、指定デバイスをリスナとして指定し、コマンドモードのまま終了します。(データ送信処理は起動しません)
- 機器アドレステーブルの先頭に-1 を格納すると、マルチラインメッセージの送出を行わずにデータモードに遷移してデータ送信を開始します。
- デリミタは関数内で自動的に付加されるので、予めデータに付加しておく必要はありません。
- 送信完了メッセージはデリミタ送出後にポストされます。

【使用例】

●C 言語

```
int Ret;
int BoardNo;
int AdrsTbl[4];
DWORD SendLen;
char SendDat[ ] = "0123456789";

BoardNo = 0;
AdrsTbl[0] = 2;
AdrsTbl[1] = -1;  // 終端コードが必要です
SendLen = strlen(SendDat);
Ret = PciGpibExMastSendData( BoardNo, &AdrsTbl[0], SendLen, SendDat, WM_NULL );
```

●Visual Basic

```
Dim Ret As Long
Dim BoardNo As Long
Dim AdrsTbl(1) As Long
Dim Data As String
Dim SendLen As Long

BoardNo = 0
AdrsTbl(0) = 2
AdrsTbl(1) = -1  ' 終端コードが必要です
Data = StrConv("0123456789", vbFromUnicode)
SendLen = LenB(Data)
Data = StrConv(Data, vbUnicode)
Ret = PciGpibExMastSendData( BoardNo, AdrsTbl(0), SendLen, Data, 0 )
```

●Delphi

```
var
    Ret : Integer;
    BoardNo : Integer;
    AdrsTbl : Array[0..4] of Integer;
    SendLen : Cardinal;
    SendData : pChar;

BoardNo := 0;
AdrsTbl[0] := 2;
AdrsTbl[1] := -1;
SendData := StrNew('0123456789');
SendLen := 10;
Ret := PciGpibExMastSendData( BoardNo, @AdrsTbl[0], SendLen, SendData, WM_NULL );
```

指定した機器へ文字列"0123456789"のデータ送信を実行します。

36. PciGpibExMastRecvData

【機能】

指定した機器からデータを受信します。

【書式】

●C 言語

```
int PciGpibExMastRecvData (
    int       BoardNo,
    int*      AdrsTbl,
    DWORD*    Length,
    void*     Buffer,
    UINT      Msg
);
```

●Visual Basic

バイト型/数値型変数のデータ受信の場合

```
Declare Function PciGpibExMastRecvData Lib "gpc4304.dll"( _
    ByVal  BoardNo   As Long,    _
    ByRef  AdrsTbl   As Long,    _
    ByRef  Length    As Long,    _
    ByRef  Buffer    As Any,     _
    ByVal  Msg       As Long     _
) As Long
```

文字列型変数のデータ受信の場合

```
Declare Function PciGpibExMastRecvData Lib "gpc4304.dll"( _
    ByVal  BoardNo   As Long,     _
    ByRef  AdrsTbl   As Long,     _
    ByRef  Length    As Long,     _
    ByVal  Buffer    As String,   _
    ByVal  Msg       As Long      _
) As Long
```

非同期データ受信の場合

```
Declare Function PciGpibExMastRecvData Lib "gpc4304.dll"( _
    ByVal  BoardNo   As Long,    _
    ByRef  AdrsTbl   As Long,    _
    ByRef  Length    As Long,    _
    ByVal  Buffer    As Any,     _
    ByVal  Msg       As Long     _
) As Long
```

●Delphi

```
function PciGpibExMastRecvData (
           BoardNo     : Integer;
           AdrsTbl     : Pointer;
  var      Length      : Cardinal;
           Buffer      : Pointer;
           Msg         : Cardinal
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

AdrsTbl

機器のアドレステーブルへのポインタ（参照渡し）を指定します。
アドレステーブルへの終端には必ず-1 を設定します。

Length

受信バッファのバイトサイズが格納されている変数へのポインタ（参照渡し）を指定します。
本関数呼び出し後、実際に受信したバイトサイズが格納されます。(同期入出力時)
受信バッファの領域は、必ず「受信データ長」＋「デリミタ」分の領域を確保するようにしてください。

Buffer

受信バッファ領域へのポインタ（参照渡し）を指定します。
※ Visual Basic の場合には、API の Declare Function の宣言がバイト型/数値型と文字列型とでは異なります。
※ Visual Basic では、文字列変数として固定長文字列を指定してください。
例: Dim RecvBuffer As String * 32 など
※ Visual Basic で非同期送信を行う場合は、GlobalAlloc 関数などを用いて割り当てたメモリ領域を指定してください。詳しくはサンプルプログラムをご参照ください。

Msg

非同期送信完了時にアプリケーションに通知されるメッセージを指定します。
同期動作時には WM_NULL もしくは 0 を指定してください

【戻り値】

正常終了しますと、以下の値を返します。

戻り値	意味
2	EOIを検出して受信が完了しました。
1	指定された受信データ数に達して受信が完了しました。
0	デリミタを検出して受信が完了しました。

上記以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

- 本関数はコントローラ・イン・チャージでのみ使用できます。
- 同期入出力時には、タイムアウト発生もしくは何らかの要因により処理が終了した場合において、本関数の処理を終了します。
- 非同期入出力時には、本関数はデータ受信開始のみ行い、データ受信完了を待たずに本関数の処理を終了します。データ受信完了の確認は、PciGpibExGetStatus 関数において入出力完了を監視、または、アプリケーションに通知されるメッセージにより行います。その時点で、本関数を再度呼び出すことにより、受信ステータスおよび受信データ長の取得を行います。
- 本関数を非同期入出力完了後に呼び出した場合、受信ステータスおよび受信データ長を取得して処理を終了します。
  受信ステータスおよび受信データ長を取得後は、再度本関数により受信処理が行われるようになります。
- 機器からの応答がタイムアウト時間を経過しても無い場合には、タイムアウトエラーとして転送中断が行われます。
- 受信データ長として 0 を指定すると、マルチラインメッセージとして本インタフェースモジュールをリスナ、最初の指定デバイスをトーカ、2 番目以降のデバイスをリスナとして指定し、コマンドモードのまま終了します。(データ受信処理は起動しません)
- 機器アドレステーブルに-1 を指定するとマルチラインメッセージの送出を行わずにデータモードに遷移してデータ受信を開始します。
- 受信完了メッセージはデリミタ受信後にポストされます。

【使用例】

●C 言語

```
int Ret;
int BoardNo;
int AdrsTbl[4];
DWORD RecvLen;
char RecvBuf[64];

BoardNo = 0;
AdrsTbl[0] = 2;
AdrsTbl[1] = -1;                // 終端コードが必要です。
RecvLen = 64;                      // 受信バッファの Max サイズを指定する
Ret = PciGpibExMastRecvData( BoardNo, &AdrsTbl[0], &RecvLen, RecvBuf, WM_NULL );
RecvBuf[RecvLen] = '¥0';  // 受信完了時には、RecvLen 変数には実際に
                          // 受信したデータのデータ長が格納される
printf("受信データ %s, 受信データ長 %d¥n", RecvBuf, RecvLen);
```

●Visual Basic

```
Public RecvBuf As String * 64  ' 受信バッファ領域は、標準モジュール内( .BAS ファイル)
                               ' で Public 宣言する

Dim Ret As Long
Dim BoardNo As Long
Dim AdrsTbl(1) As Long
Dim RecvLen As Long

BoardNo = 0
AdrsTbl(0) = 2
AdrsTbl(1) = -1     ' 終端コードが必要です
RecvLen = 64          ' 受信バッファの Max サイズを指定する
Ret = PciGpibExMastRecvData( BoardNo, AdrsTbl(0), RecvLen, RecvBuf, 0 )
```

●Delphi

```
var
    Ret : Integer;
    BoardNo : Integer,
    AdrsTbl : Array[0..4] of Integer;
    RecvLen : Cardinal;
    RecvBuf : Array[0..64] of Char;

BoardNo := 0;
AdrsTbl[0] := 2;
AdrsTbl[1] := -1;
RecvLen := 64;

Ret := PciGpibExMastRecvData( BoardNo, @AdrsTbl[0], RecvLen, @RecvBuf[0], WM_NULL );
```

指定した機器からのデータ受信の実行を行います。

37. PciGpibExMastSendFile

【機能】

ファイルからデータを読み込み、GP-IB バス上に送信します。

【書式】

●C 言語

```
int PciGpibExMastSendFile (
    int       BoardNo,
    int*      AdrsTbl,
    char*     FileName
);
```

●Visual Basic

```
Declare Function PciGpibExMastSendFile Lib "gpc4304.dll"( _
    ByVal  BoardNo     As Long,     _
    ByRef  AdrsTbl     As Long,     _
    ByVal  FileName    As String    _
) As Long
```

●Delphi

```
function PciGpibExMastSendFile (
        BoardNo     : Integer;
        AdrsTbl     : Pointer;
        FileName    : String
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

AdrsTbl

機器のアドレステーブルへのポインタ（参照渡し）を指定します。
アドレステーブルへの終端には必ず-1 を設定します。

FileName

送信データが格納されたファイル名を指定します。
ファイル名の指定はフルパスで記述する必要があります。
パスの記述がない場合には、カレントフォルダのファイル指定となります。

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】
- 本関数はコントローラ・イン・チャージでのみ使用できます。
- ファイルの内容は加工されずにそのままの形でバスに送出されます。
- ファイルの最後には、送信デリミタが付加されます。そのため、CRLF 等のデリミタを指定していた場合、ファイルの最後に CRLF が付加されてしまいます。従ってファイル転送等に使用する場合においては、送信デリミタには必ず EOI のみを指定してください。
- ファイルはバイナリデータとしてオープンされ読み込まれます。また、オープンされたファイルは読み込み終了後、ただちにクローズされます。
- 本関数では非同期入出力設定がされていても、非同期処理は行われず、同期して処理されます。

【使用例】

●C 言語

```
int Ret;
int AdrsTbl[2];

AdrsTbl[0] = 2;
AdrsTbl[1] = -1;
Ret = PciGpibExMastSendFile( 0, AdrsTbl, "test.dat" );
```

●Visual Basic

```
Dim Ret As Long
Dim AdrsTbl(1) As Long

AdrsTbl(0) = 2
AdrsTbl(1) = -1
Ret = PciGpibExMastSendFile( 0, AdrsTbl(0), "test.dat" )
```

●Delphi

```
var
  Ret : Integer;
  AdrsTbl : Array[0..4] of Integer;

AdrsTbl[0] := 2;
AdrsTbl[1] := -1;
Ret := PciGpibExMastSendFile( 0, @AdrsTbl [0], 'test.dat' );
```

ファイル名"test.dat"のファイルを読み込み、GP-IB バス上に送出します。

38. PciGpibExMastRecvFile

【機能】

GP-IB バスから受信したデータをファイルに書き込みます。

【書式】

●C 言語

```
int PciGpibExMastRecvFile (
  int      BoardNo,
  int*     AdrsTbl,
  DWORD*   Length,
  char*    FileName
);
```

●Visual Basic

```
Declare Function PciGpibExMastRecvFile Lib "gpc4304.dll"( _
  ByVal  BoardNo     As Long,     _
  ByVal  AdrsTbl     As Long,     _
  ByRef  Length      As Long,     _
  ByVal  FileName    As String    _
) As Long
```

●Delphi

```
function PciGpibExMastRecvFile (
         BoardNo    : Integer;
         AdrsTbl    : Pointer;
  var    Length     : Cardinal;
         FileName   : String
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

AdrsTbl

機器のアドレステーブルへのポインタ（参照渡し）を指定します。
アドレステーブルへの終端には必ず-1 を設定します。

Length

受信バッファのバイトサイズが格納されている変数へのポインタ（参照渡し）を指定します。
本関数呼び出し後、実際に受信したバイトサイズが格納されます。

FileName

受信データを格納するファイル名を指定します。
ファイル名の指定はフルパスで記述する必要があります。
パスの記述がない場合には、カレントフォルダのファイル指定となります。

【戻り値】

正常終了しますと、以下の値を返します。

戻り値	意味
2	EOIを検出して受信が完了しました。
1	指定された受信データ数に達して受信が完了しました。
0	デリミタを検出して受信が完了しました。

上記以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

- 本関数はコントローラ・イン・チャージでのみ使用できます。
- バスから読み込まれたデータに受信デリミタが含まれていた場合、その時点で受信処理を終了します。従ってファイル転送等に使用する場合において、ファイル中に CRLF 等のコードが含まれている場合には、受信デリミタには必ず EOI のみを指定してください。
- ファイルはバスから読み込まれたデータがそのままの形式で書き込まれます。
- オープンされたファイルは書き込み終了後、ただちにクローズされます。既にファイルが存在していた場合は、上書きされます。
- 本関数では非同期入出力設定がされていても、非同期処理は行われず、同期して処理されます。
- 受信バッファサイズ(Length)の指定はデリミタサイズも含めて充分余裕を持って指定してください。

【使用例】

●C 言語

```
int Ret;
int AdrsTbl[2];
DWORD Length;

AdrsTbl[0] = 2;
AdrsTbl[1] = -1;
Length = 64;
Ret = PciGpibExMastRecvFile( 0, AdrsTbl, Length, "test.dat" );
```

●Visual Basic

```
Dim Ret As Long
Dim AdrsTbl(1) As Long
Dim Length As Long

AdrsTbl(0) = 2
AdrsTbl(1) = -1
Length = 64
Ret = PciGpibExMastRecvFile( 0, AdrsTbl(0), Length, "test.dat" );
```

●Delphi

```
var
  Ret : Integer;
  AdrsTbl : Array[0..4] of Integer;
  Length : Cardinal;

AdrsTbl[0] := 2;
AdrsTbl[1] := -1;
Length := 64;
Ret := PciGpibExMastRecvFile( 0, @AdrsTbl[0], Length, 'test.dat' );
```

データ受信を実行し、ファイル"test.dat"に受信したデータを書き込みます。

39. PciGpibExSlavSendData

【機能】

データを送信します。
本関数を実行する場合には、コントローラよりトーカ指定済、ATN 信号がデアサート(信号無効)状態となっている必要があります。

【書式】

●C 言語

```
int PciGpibExSlavSendData (
  int       BoardNo,
  DWORD     Length,
  void*     Buffer,
  UINT      Msg
);
```

●Visual Basic

バイト型/数値型変数のデータ送信の場合

```
Declare Function PciGpibExSlavSendData Lib "gpc4304.dll"( _
  ByVal  BoardNo   As Long,    _
  ByVal  Length    As Long,    _
  ByRef  Buffer    As Any,     _
  ByVal  Msg       As Long     _
) As Long
```

文字列型変数のデータ送信の場合

```
Declare Function PciGpibExSlavSendData Lib "gpc4304.dll"( _
  ByVal  BoardNo   As Long,     _
  ByVal  Length    As Long,     _
  ByVal  Buffer    As String,   _
  ByVal  Msg       As Long      _
) As Long
```

非同期データ送信の場合

```
Declare Function PciGpibExSlavSendData Lib "gpc4304.dll"( _
  ByVal  BoardNo   As Long,    _
  ByVal  Length    As Long,    _
  ByVal  Buffer    As Any,     _
  ByVal  Msg       As Long     _
) As Long
```

●Delphi
```
function PciGpibExSlavSendData (
    BoardNo  : Integer;
    Length   : Cardinal;
    Buffer   : Pointer;
    Msg      : Cardinal
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

Length

送信データ長のバイトサイズを指定します。

Buffer

送信バッファ領域へのポインタ（参照渡し）を指定します。

※Visual Basic の場合には、API の Declare Function の宣言がバイト型/数値型と文字列型とでは異なります。

※Visual Basic で非同期送信を行う場合は、GlobalAlloc 関数などを用いて割り当てたメモリ領域を指定してください。詳しくはサンプルプログラムをご参照ください。

Msg

非同期送信完了時にアプリケーションに通知されるメッセージを指定します。
同期動作時には WM_NULL もしくは 0 を指定してください

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

- 本関数はコントローラ・イン・チャージ/非コントローラ・イン・チャージの両方で使用できます。
- 同期入出力時には、タイムアウト発生もしくは何らかの要因により処理が終了した場合において、本関数の処理を終了します。
- 同期入出力時には、本関数はデータ送信開始のみ行い、データ送信完了を待たずに本関数の処理を終了します。データ送信完了の確認は、PciGpibExGetStatus 関数において入出力完了を監視、または、アプリケーションに通知されるメッセージにより行います。その時点で、本関数を再度呼び出すことにより、送信ステータスの取得を行います。
- 本関数を非同期入出力完了後に呼び出した場合、送信ステータスを取得して処理を終了します。
  送信ステータスを取得後は、再度本関数により送信処理が行われるようになります。
- 機器からの応答がタイムアウト時間を経過しても無い場合には、タイムアウトエラーとして転送中断が行われます。

- デリミタは関数内で自動的に付加されるので、予めデータに付加しておく必要はありません。
- 送信完了メッセージはデリミタ送出後にポストされます。

【使用例】

●C 言語

```
int Ret;
int BoardNo;
DWORD Len;
char SendDat[ ] = "0123456789";

BoardNo = 0;
Len = strlen(SendDat);
Ret = PciGpibExSlavSendData( BoardNo, Len, SendDat, WM_NULL );
```

●Visual Basic

```
Dim Ret As Long
Dim Data As String
Dim Len As Long
Dim BoardNo As Long

BoardNo = 0
Data = StrConv("0123456789", vbFromUnicode)
Len = LenB(Data)
Data = StrConv(Data, vbUnicode)
Ret = PciGpibExSlavSendData( BoardNo, Len, Data, 0 )
```

●Delphi

```
var
    Ret : Integer;
    BoardNo : Integer;
    SendLen : Cardinal;
    SendData : pChar;

BoardNo := 0;
SendData := StrNew('0123456789');
SendLen := 10;
Ret := PciGpibExMastSendData( BoardNo, @AdrsTbl[0], SendLen, SendData, WM_NULL );
```

文字列"0123456789"のデータ送信を実行します。

40. PciGpibExSlavRecvData

【機能】

データを受信します。
関数を実行する場合には、コントローラよりリスナ指定済、ATN 信号がデアサート(信号無効)状態となっている必要があります。

【書式】

●C 言語

```
int PciGpibExSlavRecvData (
  int      BoardNo,
  DWORD*   Length,
  void*    Buffer,
  UINT     Msg
);
```

●Visual Basic

バイト型/数値型変数のデータ受信の場合

```
Declare Function PciGpibExSlavRecvData Lib "gpc4304.dll"( _
  ByVal  BoardNo  As Long,   _
  ByRef  Length   As Long,   _
  ByRef  Buffer   As Any,    _
  ByVal  Msg      As Long    _
) As Long
```

文字列型変数のデータ受信の場合

```
Declare Function PciGpibExSlavRecvData Lib "gpc4304.dll"( _
  ByVal  BoardNo  As Long,     _
  ByRef  Length   As Long,     _
  ByVal  Buffer   As String,   _
  ByVal  Msg      As Long      _
) As Long
```

非同期データ受信の場合

```
Declare Function PciGpibExSlavRecvData Lib "gpc4304.dll"( _
  ByVal  BoardNo  As Long,   _
  ByRef  Length   As Long,   _
  ByVal  Buffer   As Any,    _
  ByVal  Msg      As Long    _
) As Long
```

●Delphi

```
function PciGpibExSlavRecvData (
        BoardNo   : Integer;
    var Length    : Cardinal;
        Buffer    : Pointer;
        Msg       : Cardinal
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

Length

受信バッファのバイトサイズが格納されている変数へのポインタ（参照渡し）を指定します。
本関数呼び出し後、実際に受信したバイトサイズが格納されます。(同期入出力時)
受信バッファの領域は、必ず「受信データ長」＋「デリミタ」分の領域を確保するようにしてください。

Buffer

受信バッファ領域へのポインタ（参照渡し）を指定します。
※ Visual Basic の場合には、API の Declare Function の宣言がバイト型/数値型と文字列型とでは異なります。
※ Visual Basic では、文字列変数として固定長文字列を指定してください。
　例: Dim RecvBuffer As String * 32 など
※ Visual Basic で非同期送信を行う場合は、GlobalAlloc 関数などを用いて割り当てたメモリ領域を指定してください。詳しくはサンプルプログラムをご参照ください。

Msg

非同期送信完了時にアプリケーションに通知されるメッセージを指定します。
同期動作時には WM_NULL もしくは 0 を指定してください

【戻り値】

正常終了しますと、以下の値を返します。

戻り値	意味
2	EOIを検出して受信が完了しました。
1	指定された受信データ数に達して受信が完了しました。
0	デリミタを検出して受信が完了しました。

上記以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

・ 本関数はコントローラ・イン・チャージ/非コントローラ・イン・チャージの両方で使用できます。
・ 同期入出力時には、タイムアウト発生もしくは何らかの要因により処理が終了した場合

において、本関数の処理を終了します。

・ 非同期入出力時には、本関数はデータ受信開始のみ行い、データ受信完了を待たずに本関数の処理を終了します。データ受信完了の確認は、PciGpibExGetStatus 関数において入出力完了を監視、または、アプリケーションに通知されるメッセージにより行います。その時点で、本関数を再度呼び出すことにより、受信ステータスおよび受信データ長の取得を行います。
・ 本関数を非同期入出力完了後に呼び出した場合、受信ステータスおよび受信データ長を取得して処理を終了します。
受信ステータスおよび受信データ長を取得後は、再度本関数により受信処理が行われるようになります。
・ 機器からの応答がタイムアウト時間を経過しても無い場合には、タイムアウトエラーとして転送中断が行われます。
・ 受信完了メッセージはデリミタ受信後にポストされます。

【使用例】

●C 言語

```
int Ret;
int BoardNo;
DWORD RecvLen;
char RecvBuf[64];

BoardNo = 0;
RecvLen = 64;                           // 受信バッファの Max サイズを指定する
Ret = PciGpibExSlavRecvData( BoardNo, &RecvLen, RecvBuf, WM_NULL );
RecvBuf[RecvLen] = '¥0';  // 受信完了時には、RecvLen 変数には実際に受信したデータの
                          // データ長が格納される
printf("受信データ %s, 受信データ長 %d¥n", RecvBuf, RecvLen);
```

●Visual Basic

```
Public RecvBuf As String * 64  ' 受信バッファ領域は、標準モジュール内( .BAS ファイル)
                               ' で Public 宣言する

Dim Ret As Long
Dim BoardNo As Long
Dim RecvLen As Long

BoardNo = 0
RecvLen = 64                           ' 受信バッファの Max サイズを指定する
Ret =  PciGpibExSlavRecvData( BoardNo, RecvLen, RecvBuf, 0 )
' 受信完了時には、RecvLen 変数には実際に受信したデータのデータ長が格納される
MsgBox Left(RecvBuf, RecvLen)  ' 受信データを表示
```

●Delphi

```
var
    Ret : Integer;
    BoardNo : Integer,
    RecvLen : Cardinal;
    RecvBuf : Array[0..64] of Char;

BoardNo := 0;
AdrsTbl[0] := 2;
AdrsTbl[1] := -1;
RecvLen := 64;

Ret := PciGpibExSlavRecvData( BoardNo, RecvLen, @RecvBuf[0], WM_NULL );
```

データ受信の実行を行います。

41.　PciGpibExSlavSendFile

【機能】

ファイルからデータを読み込み、GP-IB バス上に送信します。

【書式】

●C 言語

```
int PciGpibExSlavSendFile (
    int       BoardNo,
    char*     FileName
);
```

●Visual Basic

```
Declare Function PciGpibExSlavSendFile Lib "gpc4304.dll"( _
    ByVal  BoardNo     As Long,          _
    ByVal  FileName    As String         _
) As Long
```

●Delphi

```
function PciGpibExSlavSendFile (
        BoardNo    : Integer;
        FileName   : String
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

FileName

送信データが格納されたファイル名を指定します。
ファイル名の指定はフルパスで記述する必要があります。
パスの記述がない場合には、カレントフォルダのファイル指定となります。

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】
- 本関数はコントローラ・イン・チャージ/非コントローラ・イン・チャージの両方で使用できます。
- ファイルの内容は加工されずにそのままの形でバスに送出されます。
- ファイルの最後には、送信デリミタが付加されます。そのため、CRLF 等のデリミタを指定していた場合、ファイルの最後に CRLF が付加されてしまいます。従ってファイル転送等に使用する場合においては、送信デリミタには必ず EOI のみを指定してください。
- ファイルはバイナリデータとしてオープンされ読み込まれます。また、オープンされたファイルは読み込み終了後、ただちにクローズされます。
- 本関数では非同期入出力設定がされていても、非同期処理は行われず、同期して処理されます。

【使用例】

●C 言語
```
int Ret;
Ret = PciGpibExSlavSendFile( 0, ″test.dat″ );
```

●Visual Basic
```
Dim Ret As Long
Ret = PciGpibExSlavSendFile( 0, ″test.dat″ )
```

●Delphi
```
var
  Ret : Integer;

Ret := PciGpibExSlavSendFile( 0, ’test.dat’ );
```
ファイル名″test.dat″のファイルを読み込み、GP-IB バス上に送出します。

42. PciGpibExSlavRecvFile

【機能】

GP-IB バスから受信したデータをファイルに書き込みます。

【書式】

●C 言語

```
int PciGpibExSlavRecvFile (
  int      BoardNo,
  DWORD*   Length,
  char*    FileName
);
```

●Visual Basic

```
Declare Function PciGpibExSlavRecvFile Lib "gpc4304.dll"( _
  ByVal  BoardNo     As Long,     _
  ByRef  Length      As Long,     _
  ByVal  FileName    As String    _
) As Long
```

●Delphi

```
function PciGpibExSlavRecvFile (
       BoardNo   : Integer;
  var  Length    : Cardinal;
       FileName  : String
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

Length

受信バッファのバイトサイズが格納されている変数へのポインタ（参照渡し）を指定します。本関数呼び出し後、実際に受信したバイトサイズが格納されます。

FileName

受信データを格納するファイル名を指定します。
ファイル名の指定はフルパスで記述する必要があります。
パスの記述がない場合には、カレントフォルダのファイル指定となります。

【戻り値】

正常終了しますと、以下の値を返します。

戻り値	意味
2	EOIを検出して受信が完了しました。
1	指定された受信データ数に達して受信が完了しました。

戻り値	意味
0	デリミタを検出して受信が完了しました。

上記以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

・ 本関数はコントローラ・イン・チャージ/非コントローラ・イン・チャージの両方で使用できます。
・ バスから読み込まれたデータに受信デリミタが含まれていた場合、その時点で受信処理を終了します。従ってファイル転送等に使用する場合において、ファイル中に CRLF 等のコードが含まれている場合には、受信デリミタには必ず EOI のみを指定してください。
・ ファイルはバスから読み込まれたデータがそのままの形式で書き込まれます。
・ オープンされたファイルは書き込み終了後、ただちにクローズされます。既にファイルが存在していた場合は、上書きされます。
・ 本関数では非同期入出力設定がされていても、非同期処理は行われず、同期して処理されます。
・ 受信バッファサイズ(Length)の指定はデリミタサイズも含めて充分余裕を持って指定してください。

【使用例】

●C 言語

```
int Ret;
DWORD Length;

Length = 64;
Ret = PciGpibExSlavRecvFile( 0, &Length, ″test.dat″ );
```

●Visual Basic

```
Dim Ret As Long
Dim Length As Long

Length = 64
Ret = PciGpibExSlavRecvFile( 0, Length, ″test.dat″ );
```

●Delphi

```
var
  Ret : Integer;
  Length : Cardinal;

Length := 64;
Ret := PciGpibExSlavRecvFile( 0, Length, 'test.dat' );
```

データ受信を実行し、ファイル″test.dat″に受信したデータを書き込みます。

43. PciGpibExSlavCheckStb

【機能】

ステータス・バイトがコントローラへ通知済であるかどうか確認します。

【書式】

●C 言語

```
int PciGpibExSlavCheckStb (
    int   BoardNo
);
```

●Visual Basic

```
Declare Function PciGpibExSlavCheckStb Lib "gpc4304.dll"( _
    ByVal   BoardNo   As Long    _
) As Long
```

●Delphi

```
function PciGpibExSlavCheckStb (
      BoardNo   : Integer
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

【戻り値】

正常終了しますと、以下の値を返します。

戻り値	意味
7	ステータス・バイトはコントローラへ通知済です。
6	まだ、シリアル・ポールは行われていません。

上記以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

・ 本関数はコントローラ・イン・チャージ/非コントローラ・イン・チャージの両方にて使用可能です。

【使用例】

●C 言語

```
int Ret;
Ret = PciGpibExSlavCheckStb( 0 );
```

●Visual Basic

```
Dim Ret As Long
Ret = PciGpibExSlavCheckStb( 0 )
```

●Delphi

```
var
  Ret : Integer;

Ret := PciGpibExSlavCheckStb( 0 );
```

ステータス・バイトがコントローラへ通知済であるか否かを確認します。

44.　PciGpibExSlavSetSrq

【機能】

ステータス・バイトを設定し、SRQ 信号を送出します。

【書式】

●C 言語

```
int PciGpibExSlavSetSrq (
    int   BoardNo,
    int   Stb
);
```

●Visual Basic

```
Declare Function PciGpibExSlavSetSrq Lib "gpc4304.dll"( _
    ByVal   BoardNo     As Long,     _
    ByVal   Stb         As Long      _
) As Long
```

●Delphi

```
function PciGpibExSlavSetSrq (
      BoardNo   : Integer;
      Stb       : Integer
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

Stb

ステータスバイトを指定します。
設定できるステータス・バイト情報は、ビット 6 を除く 7 ビットの情報です。
ビット 6 には 0 を指定してください。

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
STB7	0	STB5	STB4	STB3	STB2	STB1	STB0

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

・ 本関数はコントローラ・イン・チャージ/非コントローラ・イン・チャージの両方にて使用可能です。

【使用例】

●C 言語

```
int Ret;
Ret = PciGpibExSlavSetSrq( 0, 0x01 );
```

●Visual Basic

```
Dim Ret As Long
Ret = PciGpibExSlavSetSrq( 0, &h01 )
```

●Delphi

```
var
  Ret : Integer;

Ret := PciGpibExSlavSetSrq( 0, $01 );
```

ステータス・バイト値 1 を設定し、SRQ 信号を送出します。

45. PciGpibExSlavSetIst

【機能】

パラレルポールにおけるステータスビット(ist ビット)のセット/クリアを行います。

【書式】

●C 言語

```
int PciGpibExSlavSetIst (
    int   BoardNo,
    int   Mode
);
```

●Visual Basic

```
Declare Function PciGpibExSlavSetIst Lib "gpc4304.dll"( _
    ByVal   BoardNo   As Long,    _
    ByVal   Mode      As Long     _
) As Long
```

●Delphi

```
function PciGpibExSlavSetIst (
     BoardNo   : Integer;
     Mode      : Integer
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

Mode

パラレルポールステータス設定を指定します。

設定できるステータス・バイト情報は、ビット 6 を除く 7 ビットの情報です。

パラメータ	説明
0 以外	ist(Individual Status)ビットをセット(1)します。
0	ist(Individual Status)ビットをクリア(0)します。

【戻り値】

正常終了した場合は、0 が返されます。

0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

本関数はコントローラ・イン・チャージ/非コントローラ・イン・チャージの両方にて使用可能です。

【使用例】

●C 言語

```
int Ret;
Ret = PciGpibExSlavSetIst( 0, 1 );
```

●Visual Basic

```
Dim Ret As Long
Ret = PciGpibExSlavSetIst( 0, 1 )
```

●Delphi

```
var
  Ret : Integer;

Ret := PciGpibExSlavSetIst( 0, 1 );
```

ist ビットをセット(1)します。

46. PciGpibExSlavSetPp2

【機能】

ローカルコンフィグレーション(pp2)における、パラレルポール応答モードを設定します。

【書式】

●C 言語

```
int PciGpibExSlavSetPp2 (
    int   BoardNo,
    int   Mode
);
```

●Visual Basic

```
Declare Function PciGpibExSlavSetPp2 Lib "gpc4304.dll"( _
    ByVal   BoardNo    As Long,    _
    ByVal   Mode       As Long     _
) As Long
```

●Delphi

```
function PciGpibExSlavSetPp2 (
     BoardNo  : Integer;
     Mode     : Integer
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

Mode

パラレルポール応答モードを指定します。

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
0	0	0	0	S	p3	p2	p1

p3	p2	p1	DIO ライン 1～8 の指定
0	0	0	DIO1 で応答
0	0	1	DIO2 で応答
0	1	0	DIO3 で応答
0	1	1	DIO4 で応答
1	0	0	DIO5 で応答
1	0	1	DIO6 で応答
1	1	0	DIO7 で応答
1	1	1	DIO8 で応答

S	極性の指定
0	機器の ist ビットがクリア(0)のとき、割り当てた応答 DIO ラインをアサートします。
1	機器の ist ビットがセット(1)のとき、割り当てた応答 DIO ラインをアサートします。

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

・ 本関数はコントローラ・イン・チャージ/非コントローラ・イン・チャージの両方にて使用可能です。

【使用例】

●C 言語

```
int Ret;
Ret = PciGpibExSlavSetPp2( 0, 0x09 );
```

●Visual Basic

```
Dim Ret As Long
Ret = PciGpibExSlavSetPp2( 0, &H9 )
```

●Delphi

```
var
  Ret : Integer;

Ret := PciGpibExSlavSetPp2( 0, $09 );
```

パラレルポーリング実行時に、ist ビットが 1 ならば DIO ライン 2 をアサートする設定を行います。

47. PciGpibExWaitTimer

【機能】

指定した時間待ちます。

【書式】

●C 言語

```
int PciGpibExWaitTimer (
   DWORD   Timeout
);
```

●Visual Basic

```
Declare Function PciGpibExWaitTimer Lib "gpc4304.dll" ( _
  ByVal   Timeout    As Long     _
) As Long
```

●Delphi

```
function PciGpibExWaitTimer (
     Timeout   : Cardinal
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

Timeout

待ち時間を 100 ミリ秒単位で指定します。
(設定範囲:0～FFFFFFFFh)

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4 戻り値一覧』をご参照ください。

【備考】

・ 本関数を呼び出し後､指定時間を経過するまでアプリケーションプログラムには制御が戻りません。

【使用例】

●C 言語

```
int Ret;
Ret = PciGpibExWaitTimer( 10 );
```

●Visual Basic

```
Dim Ret As Long
Ret = PciGpibExWaitTimer( 10 )
```

●Delphi

```
var
  Ret : Integer;

Ret := PciGpibExWaitTimer( 10 );
```

1秒間待ちます。

48. PciGpibExStartTimer

【機能】

汎用タイマのスタートを行います。

【書式】

●C 言語

```
int PciGpibExStartTimer (void);
```

●Visual Basic

```
Declare Function PciGpibExStartTimer Lib "gpc4304.dll"( ) As Long
```

●Delphi

```
function PciGpibExStartTimer : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

なし

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

・ 周期は 100ms、32bit カウンタです。
・ 本関数はマルチメディアタイマを使用しているので、OS の負荷により数 ms～数 10ms の誤差が出る場合があります。

【使用例】

●C 言語

```
int Ret;
Ret = PciGpibExStartTimer( );
```

●Visual Basic

```
Dim Ret As Long
Ret = PciGpibExStartTimer( )
```

●Delphi

```
var
  Ret : Integer;

Ret := PciGpibExStartTimer( );
```

汎用タイマのスタートを行います。

49. PciGpibExClearTimer

【機能】

汎用タイマの現在値をクリアして、再スタートを行います。

【書式】

●C 言語

```
int PciGpibExClearTimer (void);
```

●Visual Basic

```
Declare Function PciGpibExClearTimer Lib "gpc4304.dll"( ) As Long
```

●Delphi

```
function PciGpibExClearTimer : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

なし

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4 戻り値一覧』をご参照ください。

【使用例】

●C 言語

```
int Ret;
Ret = PciGpibExClearTimer( );
```

●Visual Basic

```
Dim Ret As Long
Ret = PciGpibExClearTimer( )
```

●Delphi

```
var
  Ret : Integer;

Ret := PciGpibExClearTimer( );
```

汎用タイマの再スタートを行います。

50. PciGpibExReadTimer

【機能】

汎用タイマの現在値を取得します。

【書式】

●C 言語

```
int PciGpibExReadTimer (
   DWORD*        TimerValue
);
```

●Visual Basic

```
Declare Function PciGpibExReadTimer Lib "gpc4304.dll" ( _
  ByRef  TimerValue   As Long   _
) As Long
```

●Delphi

```
function PciGpibExReadTimer (
  var  TimerValue   : Cardinal              _
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

TimerValue

現在のタイマ値を格納する変数のポインタ（参照渡し）を指定します。

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【使用例】

●C 言語

```
int Ret;
DWORD TimeVal;
Ret = PciGpibExReadTimer( &TimeVal );
```

●Visual Basic

```
Dim Ret As Long
Dim TimeVal As Long
Ret = PciGpibExReadTimer( TimeVal )
```

●Delphi

```
var
  Ret : Integer;
  TimeVal : Cardinal;

Ret := PciGpibExReadTimer( TimeVal );
```

汎用タイマの読み出しを行います。

51. PciGpibExStopTimer

【機能】

汎用タイマを停止させます。

【書式】

●C 言語

```
int PciGpibExStopTimer (void);
```

●Visual Basic

```
Declare Function PciGpibExStopTimer Lib "gpc4304.dll"( ) As Long
```

●Delphi

```
function PciGpibExStopTimer : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

なし

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【使用例】

●C 言語

```
int Ret;
Ret = PciGpibExStopTimer( );
```

●Visual Basic

```
Dim Ret As Long
Ret = PciGpibExStopTimer( )
```

●Delphi

```
var
  Ret : Integer;

Ret := PciGpibExStopTimer( );
```

汎用タイマを停止させます。

52.　PciGpibExSetSrqEvent

【機能】

SRQ コールバックイベントを登録します。

【書式】

●C 言語(x86)

```
int PciGpibExSetSrqEvent (
    Int              BoardNo,
    LPSRQCALLBACK    SrqProc,
    DWORD            dwUser
);
```

●C 言語(x64)

```
int PciGpibExSetSrqEvent (
    int              BoardNo,
    LPSRQCALLBACK    SrqProc,
    PVOID            dwUser
);
```

●Visual Basic

```
Declare Function PciGpibExSetSrqEvent Lib "gpc4304.dll"( _
    ByVal    BoardNo    As Long,    _
    ByVal    SrqProc    As Long,    _
    ByVal    dwUser     As Long     _
) As Long
```

●Delphi

```
Function PciGpibExSetSrqEvent (
      BoardNo     : Cardinal;
      SrqProc     : FARPROC;
      DwUser      : DWORD
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

SrqProc

コールバック関数のポインタ（参照渡し）を指定します。
『4.3　コールバック関数』をご参照ください。

dwUser

コールバック関数に渡す任意のデータを指定します。

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

コールバック関数の書式は以下の通りです (C言語(x86))。『4.3　コールバック関数』もご参照ください。

```
void CALLBACK EventProc (
    Int       BoardNo,
    DWORD     dwUser
);
```

【使用例】

●C 言語(x86)

```
void CALLBACK OnSrqProc (int BoardNo, DWORD dwUser) {
                ・
                ・
                ・
}

int Ret;
Ret = PciGpibExSetSrqEvent( 0, (PLPSRQCALLBACK)OnSrqProc, 100 );
```

●C 言語(x64)

```
void CALLBACK OnSrqProc (int BoardNo, PVOID dwUser) {
                ・
                ・
                ・
}

int Ret;
Ret = PciGpibExSetSrqEvent( 0, (PLPSRQCALLBACK)OnSrqProc, 100 );
```

●Visual Basic

コールバックルーチンは PciGpibExSetSrqEvent 関数の呼び出しを行うプロジェクト内の標準モジュールの中に記述しなければなりません。

・ PciGpibExSetSrqEvent 関数の引数パラメータでプロシージャのアドレスを渡す為に AddressOf 演算子を使用します。
(PciGpibExSetSrqEvent 関数の使用例を参照してください。)
・ AddressOf 演算子を使うと、プロシージャからの戻り値ではなく、プロシージャ自体のアドレスが、ダイナミック リンク ライブラリ (DLL)の PciGpibExSetSrqEvent 関数に渡されます。

```
Sub OnSrqProc(BoardNo As Long, dwUser As Long)
              ' ・
              ' ・ 割り込みイベントに対応する処理を記述します。
              ' ・
End Sub

Dim Ret As Long
Ret = PciGpibExSetSrqEvent( 0, AddressOf OnSrqProc, 100 )
```

●Delphi

```
procedure OnSrqProc(BoardNo : Integer, dwUser: DWORD); stdcall;
var
      // 変数定義

begin
             // ・
             // ・ 割り込みイベントに対応する処理を記述します。
             // ・
end;

var
  Ret : Integer;

Ret := PciGpibExSetSrqEvent( 0, OnSrqProc, 100 );
```

ボード番号 0 の GP-IB インタフェースモジュールに対して SRQ コールバック関数 lpOnSrqProc を登録します。

53. PciGpibExWaitSrqEvent

【機能】

SRQ コールバックイベントを待ちます。

【書式】

●C 言語

```
int PciGpibExSetSrqEvent (
    int      BoardNo,
    ULONG    Timeout
);
```

●Visual Basic

```
Declare Function PciGpibExSetSrqEvent Lib "gpc4304.dll"( _
    ByVal    BoardNo     As         _
                         Long,
    ByVal    Timeout     As Long    _
) As Long
```

●Delphi

```
function PciGpibExSetSrqEvent (
      BoardNo   : Cardinal;
      Timeout   : Cardinal
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

Timeout

タイムアウト時間をミリ秒単位で指定します。
0 を指定した場合は、状態を調べてからすぐに制御を戻します

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【使用例】

●C 言語

```
int Ret, BoardNo;
BoardNo = 0;
Ret = PciGpibExWaitSrqEvent( BoardNo, 5000 );
```

●Visual Basic

```
Dim Ret As Long
Dim BoardNo As Long
BoardNo = 0
Ret = PciGpibExWaitSrqEvent( BoardNo, 5000 )
```

●Delphi

```
var
  Ret : Integer;
  BoardNo : Cardinal;

BoardNo := 0;
Ret := PciGpibExWaitSrqEvent( BoardNo, 5000 );
```

SRQ コールバックイベント待ちをします。(タイムアウト 5 秒)

54. PciGpibExKillSrqEvent

【機能】

SRQ コールバックイベントの登録を解除します。

【書式】

●C 言語

```
int PciGpibExKillSrqEvent (
   int   BoardNo
);
```

●Visual Basic

```
Declare Function PciGpibExKillSrqEvent Lib "gpc4304.dll"( _
   ByVal   BoardNo  As Long   _
) As Long
```

●Delphi

```
function PciGpibExKillSrqEvent (
     BoardNo  : Cardinal
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

【戻り値】

正常終了した場合は、0 が返されます。

0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【使用例】

●C 言語

```
int Ret, BoardNo;
BoardNo = 0;
Ret = PciGpibExKillSrqEvent( BoardNo );
```

●Visual Basic

```
Dim Ret As Long
Dim BoardNo As Long
BoardNo = 0
Ret = PciGpibExKillSrqEvent( BoardNo )
```

●Delphi

```
var
  Ret : Integer;
  BoardNo : Cardinal;

BoardNo := 0;
Ret := PciGpibExKillSrqEvent( BoardNo );
```

SRQ コールバックイベント登録を解除します。

55. PciGpibExMastSendDelay

【機能】

指定した機器に対して、データを送信します。
バス・コマンド(マルチライン・インタフェース・メッセージ)の送出する間隔を指定することができます。

【書式】

●C 言語

```
int PciGpibExMastSendDelay (
    int       BoardNo,
    int*      AdrsTbl,
    DWORD     Length,
    void*     Buffer,
    UINT      Msg,
    int       Delay
);
```

●Visual Basic

バイト型/数値型変数のデータ送信の場合

```
Declare Function PciGpibExMastSendDelay Lib "gpc4304.dll"( _
    ByVal  BoardNo   As Long,         _
    ByRef  AdrsTbl   As Long,         _
    ByVal  Length    As Long,         _
    ByRef  Buffer    As Any,          _
    ByVal  Msg       As Long,         _
    ByVal  Delay     As Long          _
) As Long
```

文字列型変数のデータ送信の場合

```
Declare Function PciGpibExMastSendDelay Lib "gpc4304.dll"( _
    ByVal  BoardNo   As Long,     _
    ByRef  AdrsTbl   As Long,     _
    ByVal  Length    As Long,     _
    ByVal  Buffer    As String,   _
    ByVal  Msg       As Long,     _
    ByVal  Delay     As Long      _
) As Long
```

非同期データ送信の場合

```
Declare Function PciGpibExMastSendDelay Lib "gpc4304.dll"( _
  ByVal  BoardNo   As Long,       _
  ByRef  AdrsTbl   As Long,       _
  ByVal  Length    As Long,       _
  ByVal  Buffer    As Any,        _
  ByVal  Msg       As Long,       _
  ByVal  Delay     As Long        _
) As Long
```

●Delphi

```
function PciGpibExMastSendDelay (
      BoardNo     : Integer;
      AdrsTbl     : Pointer;
      Length      : Cardinal;
      Buffer      : Pointer;
      Msg         : Cardinal;
      Delay       : Integer
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

AdrsTbl

機器のアドレステーブルへのポインタ（参照渡し）を指定します。
アドレステーブルへの終端には必ず-1 を設定します。

Length

送信データ長のバイトサイズを指定します。

Buffer

送信バッファ領域へのポインタ（参照渡し）を指定します。

※ Visual Basic の場合には、API の Declare Function の宣言がバイト型/数値型と文字列型とでは異なります。

※ Visual Basic で非同期送信を行う場合は、GlobalAlloc 関数などを用いて割り当てたメモリ領域を指定してください。詳しくはサンプルプログラムをご参照ください。

Msg

非同期送信完了時にアプリケーションに通知されるメッセージを指定します。
同期動作時には WM_NULL もしくは 0 を指定してください

Delay

バス・コマンドの送出間隔を ms 単位で設定します。0～FFFFFFFFh が指定可能です。

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

- 本関数はコントローラ・イン・チャージでのみ使用できます。
- 同期入出力時には、タイムアウト発生もしくは何らかの要因により処理が終了した場合において、本関数の処理を終了します。
- 非同期入出力時は、本関数はデータ送信開始のみ行い、データ送信完了を待たずに本関数の処理を終了します。データ送信完了の確認は、PciGpibExGetStatus 関数において入出力完了を監視、または、アプリケーションに通知されるメッセージにより行います。その時点で、本関数を再度呼び出すことにより、送信ステータスの取得を行います。
- 本関数を非同期入出力完了後に呼び出した場合、送信ステータスを取得して処理を終了します。
  送信ステータスを取得後は、再度本関数により送信処理が行われるようになります。
- 機器からの応答がタイムアウト時間を経過しても無い場合には、タイムアウトエラーとして転送中断が行われます。
- 送信データ長として 0 を指定すると、マルチラインメッセージとして本インタフェースモジュールをトーカ、指定デバイスをリスナとして指定し、コマンドモードのまま終了します。(データ送信処理は起動しません)
- 機器アドレステーブルの先頭に-1 を格納すると、マルチラインメッセージの送出を行わずにデータモードに遷移してデータ送信を開始します。
- デリミタは関数内で自動的に付加されるので、予めデータに付加しておく必要はありません。
- 送信完了メッセージはデリミタ送出後にポストされます。
- バス・コマンド間隔の測定には OS のタイマを使用しているため、実際の間隔が 10ms 単位に切り上げられる（例えば 5ms と指定した場合、実際の間隔は 10ms となる）場合があります。

【使用例】

●C 言語

```
int Ret;
int BoardNo;
int AdrsTbl[4];
DWORD SendLen;
char SendDat[ ] = "0123456789";

BoardNo = 0;
AdrsTbl[0] = 2;
AdrsTbl[1] = -1;  // 終端コードが必要です
SendLen = strlen(SendDat);
Ret = PciGpibExMastSendDelay( BoardNo, &AdrsTbl[0], SendLen, SendDat, WM_NULL, 50 );
```

●Visual Basic

```
Dim Ret As Long
Dim BoardNo As Long
Dim AdrsTbl(1) As Long
Dim Data As String
Dim SendLen As Long

BoardNo = 0
AdrsTbl(0) = 2
AdrsTbl(1) = -1  ' 終端コードが必要です
Data = StrConv("0123456789", vbFromUnicode)
SendLen = LenB(Data)
Data = StrConv(Data, vbUnicode)
Ret = PciGpibExMastSendDelay(BoardNo, AdrsTbl(0), SendLen, Data, 0, 50)
```

●Delphi

```
var
    Ret : Integer;
    BoardNo : Integer;
    AdrsTbl : Array[0..4] of Integer;
    SendLen : Cardinal;
    SendData : pChar;

BoardNo := 0;
AdrsTbl[0] := 2;
AdrsTbl[1] := -1;
SendData := StrNew('0123456789');
SendLen := 10;
Ret := PciGpibExMastSendDelay(BoardNo, @AdrsTbl[0], SendLen, SendData, WM_NULL,
50 );
```

指定した機器へ 50ms 間隔でバスコマンド送出後、文字列"0123456789"のデータ送信を実行します。

56.　PciGpibExMastRecvDelay

【機能】

指定した機器からデータを受信します。
バス・コマンド(マルチライン・インタフェース・メッセージ)の送出する間隔を指定することができます。

【書式】

●C 言語

```
int PciGpibExMastRecvDelay (
    int        BoardNo,
    int*       AdrsTbl,
    DWORD*     Length,
    void*      Buffer,
    UINT       Msg,
    int        Delay
);
```

●Visual Basic

バイト型/数値型変数のデータ受信の場合

```
Declare Function PciGpibExMastRecvDelay Lib "gpc4304.dll"( _
    ByVal  BoardNo    As Long,    _
    ByRef  AdrsTbl    As Long,    _
    ByRef  Length     As Long,    _
    ByRef  Buffer     As Any,     _
    ByVal  Msg        As Long,    _
    ByVal  Delay      As Long     _
) As Long
```

文字列型変数のデータ受信の場合

```
Declare Function PciGpibExMastRecvDelay Lib "gpc4304.dll"( _
    ByVal  BoardNo    As Long,      _
    ByRef  AdrsTbl    As Long,      _
    ByRef  Length     As Long,      _
    ByVal  Buffer     As String,    _
    ByVal  Msg        As Long,      _
    ByVal  Delay      As Long       _
) As Long
```

非同期データ受信の場合

```
Declare Function PciGpibExMastRecvDelay Lib "gpc4304.dll"( _
  ByVal  BoardNo    As Long,    _
  ByRef  AdrsTbl    As Long,    _
  ByRef  Length     As Long,    _
  ByVal  Buffer     As Any,     _
  ByVal  Msg        As Long,    _
  ByVal  Delay      As Long     _
) As Long
```

●Delphi

```
function PciGpibExMastRecvDelay (
         BoardNo    : Integer;
         AdrsTbl    : Pointer;
  var    Length     : Cardinal;
         Buffer     : Pointer;
         Msg        : Cardinal;
         Delay      : Integer
) : Integer; stdcall; external 'GPC4304.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

AdrsTbl

機器のアドレステーブルへのポインタ（参照渡し）を指定します。
アドレステーブルへの終端には必ず-1 を設定します。

Length

受信バッファのバイトサイズが格納されている変数へのポインタ（参照渡し）を指定します。
本関数呼び出し後、実際に受信したバイトサイズが格納されます。(同期入出力時)
受信バッファの領域は、必ず「受信データ長」＋「デリミタ」分の領域を確保するようにしてください。

Buffer

受信バッファ領域へのポインタ（参照渡し）を指定します。
※ Visual Basic の場合には、API の Declare Function の宣言がバイト型/数値型と文字列型とでは異なります。
※ Visual Basic では、文字列変数として固定長文字列を指定してください。
例: Dim RecvBuffer As String * 32 など
※ Visual Basic で非同期送信を行う場合は、GlobalAlloc 関数などを用いて割り当てたメモリ領域を指定してください。詳しくはサンプルプログラムをご参照ください。

Msg

非同期送信完了時にアプリケーションに通知されるメッセージを指定します。
同期動作時には WM_NULL もしくは 0 を指定してください

Delay
バス・コマンドの送出間隔を ms 単位で設定します。0～FFFFFFFFh が指定可能です。

【戻り値】
正常終了しますと、以下の値を返します。

戻り値	意味
2	EOIを検出して受信が完了しました。
1	指定された受信データ数に達して受信が完了しました。
0	デリミタを検出して受信が完了しました。

上記以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】
- 本関数はコントローラ・イン・チャージでのみ使用できます。
- 同期入出力時には、タイムアウト発生もしくは何らかの要因により処理が終了した場合において、本関数の処理を終了します。
- 非同期入出力時には、本関数はデータ受信開始のみ行い、データ受信完了を待たずに本関数の処理を終了します。データ受信完了の確認は、PciGpibExGetStatus 関数において入出力完了を監視、または、アプリケーションに通知されるメッセージにより行います。その時点で、本関数を再度呼び出すことにより、受信ステータスおよび受信データ長の取得を行います。
- 本関数を非同期入出力完了後に呼び出した場合、受信ステータスおよび受信データ長を取得して処理を終了します。
受信ステータスおよび受信データ長を取得後は、再度本関数により受信処理が行われるようになります。
- 機器からの応答がタイムアウト時間を経過しても無い場合には、タイムアウトエラーとして転送中断が行われます。
- 受信データ長として 0 を指定すると、マルチラインメッセージとして本インタフェースモジュールをリスナ、最初の指定デバイスをトーカ、2 番目以降のデバイスをリスナとして指定し、コマンドモードのまま終了します。(データ受信処理は起動しません)
- 機器アドレステーブルに-1 を指定するとマルチラインメッセージの送出を行わずにデータモードに遷移してデータ受信を開始します。
- 受信完了メッセージはデリミタ受信後にポストされます。
- バス・コマンド間隔の測定には OS のタイマを使用しているため、実際の間隔が 10ms 単位に切り上げられる（例えば 5ms と指定した場合、実際の間隔は 10ms となる）場合があります。

【使用例】

●C 言語

```
int Ret;
int BoardNo;
int AdrsTbl[4];
DWORD RecvLen;
char RecvBuf[64];

BoardNo = 0;
AdrsTbl[0] = 2;
AdrsTbl[1] = -1;                // 終端コードが必要です。
RecvLen = 64;                     // 受信バッファの Max サイズを指定する
Ret = PciGpibExMastRecvDelay( BoardNo, &AdrsTbl[0],
                             &RecvLen, RecvBuf, WM_NULL, 50 );
RecvBuf[RecvLen] = '¥0';  // 受信完了時には、RecvLen 変数には実際に
                          // 受信したデータのデータ長が格納される
printf("受信データ %s, 受信データ長 %d¥n", RecvBuf, RecvLen);
```

●Visual Basic

```
Public RecvBuf As String * 64  ' 受信バッファ領域は、標準モジュール内( .BAS ファイル)
                               ' で Public 宣言する

Dim Ret As Long
Dim BoardNo As Long
Dim AdrsTbl(1) As Long
Dim RecvLen As Long

BoardNo = 0
AdrsTbl(0) = 2
AdrsTbl(1) = -1     ' 終端コードが必要です
RecvLen = 64          ' 受信バッファの Max サイズを指定する
Ret = PciGpibExMastRecvDelay( BoardNo, AdrsTbl(0), RecvLen, RecvBuf, 0, 50 )
```

●Delphi

```
var
    Ret : Integer;
    BoardNo : Integer,
    AdrsTbl : Array[0..4] of Integer;
    RecvLen : Cardinal;
    RecvBuf : Array[0..64] of Char;

BoardNo := 0;
AdrsTbl[0] := 2;
AdrsTbl[1] := -1;
RecvLen := 64;

Ret := PciGpibExMastRecvDelay(BoardNo, @AdrsTbl[0], RecvLen, @RecvBuf[0], WM_NULL,
50);
```

指定した機器へ 50ms 間隔でバスコマンド送出後、データ受信の実行を行います。

4.2 標準版DLL

4.2.1 関数一覧

No	関数名	機能
1	GpibOpen	インタフェースモジュールをオープンします。
2	GpibSetIfc	IFC信号を送出し、インタフェースモジュールをコントローラとします。
3	GpibSetRen	REN信号を有効にします。
4	GpibSetConfig	タイムアウト時間、デリミタ設定を変更します。
5	GpibExecDeviceTrigger	デバイストリガを実行します。
6	GpibExecDeviceClear	デバイスクリアを実行します。
7	GpibCheckSrq	SRQ信号を受信しているかを確認します。
8	GpibExecSpoll	シリアルポーリングを実行します。
9	GpibReceive	計測機器から受信を行います。
10	GpibSend	計測機器に対して送信を行います。
11	GpibClose	インタフェースモジュールをクローズします。
12	GpibSetSrqEvent	SRQコールバックイベントを登録します。
13	GpibWaitSrqEvent	SRQコールバックイベントを待ちます。
14	GpibKillSrqEvent	SRQコールバックイベント登録を解除します。

4.2.2 関数個別説明

1. GpibOpen

【機能】

オープンを行い、以後のインタフェースモジュールへのアクセスを行えるようにします。

【書式】

●C 言語

```
int GpibOpen (
  ULONG      BoardNo
);
```

●Visual Basic

```
Declare Function GpibOpen Lib "gpc43042.dll"( _
  ByVal   BoardNo    As Long          _
) As Long
```

●Delphi

```
function GpibOpen (
     BoardNo  : Cardinal
) : Integer; stdcall; external 'GPC43042.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。
インタフェースモジュール上のロータリスイッチの値がボードアクセス番号となります。
ソルコン Classembly Devices®、I/O 付きタッチパネル Classembly Devices®の GP-IB モデルをお使いの場合は、0 を指定してください。
CardBus シリーズにつきましては、CardBus ID 設定ユーティリティで設定した ID がボードアクセス番号となります。

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

コントロールパネルで次のパラメータが設定してある場合には、エラーとなります。

- 送受信デリミタとして「デリミタなし」、「NULL」、「任意の 1 文字」を指定している。
  ※デリミタは、「EOI のみ」,「CR」,「CR+EOI」,「LF」,「LF+EOI」,「CRLF」,「CRLF+EOI」の 7 種のみ使用可能です。
- 動作モードで「非コントローラモード」を指定している。

【使用例】

●C 言語

```
int Ret;
ULONG BoardNo;
BoardNo = 0;
Ret = GpibOpen( BoardNo );
```

●Visual Basic

```
Dim Ret As Integer
Dim BoardNo As Long
BoardNo = 0
Ret = GpibOpen( BoardNo )
```

●Delphi

```
var
  Ret: Integer;
  BoardNo: Cardinal;

BoardNo := 0;
Ret := GpibOpen( BoardNo );
```

インタフェースモジュールをオープンします。

2. GpibSetIfc

【機能】

IFC 信号を送出して、インタフェースモジュールをコントローラとし、以後の計測機器への制御を可能とします。

【書式】

●C 言語

```
int GpibSetIfc (
  ULONG   BoardNo
);
```

●Visual Basic

```
Declare Function GpibSetIfc Lib "gpc43042.dll"( _
  ByVal   BoardNo  As Long   _
) As Long
```

●Delphi

```
function GpibSetIfc (
     BoardNo  : Cardinal
) : Integer; stdcall; external 'GPC43042.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【使用例】

●C 言語

```
int Ret;
ULONG BoardNo;
BoardNo = 0;
Ret = GpibSetIfc( BoardNo );
```

●Visual Basic

```
Dim Ret As Long
Dim BoardNo As Long
BoardNo = 0
Ret = GpibSetIfc( BoardNo )
```

●Delphi

```
var
  Ret : Integer;
  BoardNo : Cardinal;

BoardNo := 0;
Ret := GpibSetIfc( BoardNo );
```

IFC信号を送出します。

3. GpibSetRen

【機能】

REN 信号を有効にします。

【書式】

●C 言語

```
int GpibSetRen (
  ULONG     BoardNo
);
```

●Visual Basic

```
Declare Function GpibSetRen Lib "gpc43042.dll"( _
  ByVal  BoardNo   As Long   _
) As Long
```

●Delphi

```
function GpibSetRen (
     BoardNo  : Cardinal
) : Integer; stdcall; external 'GPC43042.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

【戻り値】

正常終了した場合は、0 が返されます。

0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【使用例】

●C 言語

```
int Ret;
ULONG BoardNo;
BoardNo = 0;
Ret = GpibSetRen( BoardNo );
```

●Visual Basic

```
Dim Ret As Long
Dim BoardNo As Long
BoardNo = 0
Ret = GpibSetRen( BoardNo )
```

●Delphi

```
var
  Ret : Integer;
  BoardNo : Cardinal;

BoardNo := 0;
Ret := GpibSetRen( BoardNo );
```

REN信号を有効にします。

4. GpibSetConfig

【機能】

タイムアウト時間、デリミタ設定、GP-IB アドレス設定、その他各種設定を変更します。

【書式】

●C 言語

```
int GpibSetConfig (
   ULONG    BoardNo,
   PUCHAR   Config
);
```

●Visual Basic

```
Declare Function GpibSetConfig Lib "gpc43042.dll"( _
   ByVal   BoardNo   As Long,         _
   ByVal   Config    As String        _
) As Long
```

●Delphi

```
function GpibSetConfig (
     BoardNo   : Cardinal;
     Config    : String
) : Integer; stdcall; external 'GPC43042.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

Config

設定に関する各種情報を含んだ文字列を指定してください。
指定のなかった項目に対しては、初期値が設定されます。
設定情報の各パラメータを以下に示します。各パラメータは、スペースで区切ってください。各パラメータ文字列の中にはスペースを入れないようにしてください。
文字列の最後に NULL 文字を入れてください。

項目	文字列	内容
送受信タイムアウト	"/TMO="	送受信時に使用するタイムアウト時間(単位：100ms)を 1～65535 で設定します。 デフォルトは、コントロールパネルアプレットでの設定値が使用されます。
STB 応答時間	"/SRT="	シリアルポール時のステータスバイト応答時間(単位：100ms)を 1～65535 で設定します。 デフォルトは、コントロールパネルアプレットでの設定値が使用されます。

項目	文字列	内容	
送信デリミタ	″/SDELIM=″	送信時に使用するデリミタを設定します。 デフォルトは、コントロールパネルアプレットでの設定値が使用されます。	
		″NO″	デリミタ無し
		″EOI″	EOI
		″CR″	CR
送信デリミタ	″/SDELIM=″	″CR+EOI″	CR+EOI
		″LF″	LF
		″LF+EOI″	LF+EOI
		″CRLF″	CRLF
		″CRLF+EOI″	CRLF+EOI
受信デリミタ	″/RDELIM=″	受信時に使用するデリミタを設定します。 デフォルトは、コントロールパネルアプレットでの設定値が使用されます。	
		″NO″	デリミタ無し
		″EOI″	EOI
		″CR″	CR
		″CR+EOI″	CR+EOI
		″LF″	LF
		″LF+EOI″	LF+EOI
		″CRLF″	CRLF
		″CRLF+EOI″	CRLF+EOI
1 次アドレス	″/MA=″	1 次アドレスを設定します。範囲 0～30 デフォルトは、コントロールパネルアプレットでの設定値が使用されます。	
2 次アドレス	″/SA=″	2 次アドレスを設定します。範囲：96～126 96～126 以外の値を設定すると 2 次アドレスは無効となります。 デフォルトは、コントロールパネルアプレットでの設定値が使用されます。	
NRFD 信号ラインアサート待ち[有効／無効]	″/NRFDWAIT=″	コントローラデータ送信の最終データ送信後に GP-IB バスの NRFD 信号ラインがアサートされるまで「待つ」か「待たない」かの設定を行います。 「待ち時間」には「データ転送タイムアウト」の時間が適用されます。	
		″ON″	NRFD 信号ラインがアサートされるまで待ちます。
		″OFF″	NRFD 信号ラインのアサート待ちは無効となります。(デフォルト)

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【使用例】

●C 言語

```
int Ret;
ULONG BoardNo;
CHAR Param[] = "/TMO=100 /SDELIM=EOI";
BoardNo = 0;
Ret = GpibSetConfig( BoardNo, &Param[0] );
```

●Visual Basic

```
Dim Ret As Long
Dim BoardNo As Long
Dim Param As String
BoardNo = 0
Param = "/TMO=100 /SDELIM=EOI"
Ret = GpibSetConfig( BoardNo, Param )
```

●Delphi

```
var
  Ret : Integer;
  BoardNo : Cardinal;
  Param : String;

BoardNo := 0;
Param := '/TMO=100 /SDELIM=EOI';
Ret := GpibSetConfig( BoardNo, Param );
```

ボード番号 0 の GP-IB インタフェースモジュールの各種パラメータ情報を変更します。

5. GpibExecDeviceTrigger

【機能】

デバイストリガを実行します。

【書式】

●C 言語

```
int GpibExecDeviceTrigger (
  ULONG   BoardNo,
  PLONG   AdrsTbl
);
```

●Visual Basic

```
Declare Function GpibExecDeviceTrigger Lib "gpc43042.dll"( _
  ByVal   BoardNo  As Long,          _
  ByRef   AdrsTbl  As String         _
) As Long
```

●Delphi

```
function GpibExecDeviceTrigger (
     BoardNo  : Cardinal;
     AdrsTbl  : Pointer
) : Integer; stdcall; external 'GPC43042.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

AdrsTbl

機器のアドレステーブルを指定します。
デバイストリガを有効にする機器のアドレステーブルへのポインタ（参照渡し）を指定します。機器アドレステーブルは次の形式で指定します。

1 台の機器（アドレス 10）の場合
LONG 配列
+0[10]
+1[-1]…終端コード

2 台の機器（アドレス 10, 11）の場合
LONG 配列
+0[10]
+1[11]
+2[-1]…終端コード

※-1（終端コード）は必ず最後に付加してください。

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

本関数を実行することにより、計測機器に対してデバイストリガを実行します。デバイストリガを受信した計測機器の動作については、計測機器のマニュアルをご参照ください。

【使用例】

●C 言語

```
int Ret;
ULONG BoardNo;
LONG AdrTbl[] = { 10, -1 };  // 機器アドレステーブル
BoardNo = 0;
Ret = GpibExecDeviceTrigger( BoardNo, &AdrTbl[0] );
```

●Visual Basic

```
Dim Ret As Long
Dim BoardNo As Long
Dim AdrTbl(1) As Long
BoardNo = 0
AdrTbl(0) = 10
AdrTbl(1) = -1
Ret = GpibExecDeviceTrigger( BoardNo, AdrTbl(0) )
```

●Delphi

```
var
  Ret : Integer;
  BoardNo : Cardinal;
  AdrTbl : Array[0..1] of Integer;

BoardNo := 0;
AdrTbl[0] := 10;
AdrTbl[1] := -1;
Ret := GpibExecDeviceTrigger( BoardNo, AdrTbl[0] );
```

ボード番号0のGP-IBインタフェースモジュールからGP-IBアドレス10の機器に対して、デバイストリガを実行します。

6. GpibExecDeviceClear

【機能】

デバイスクリアを実行します。

【書式】

●C 言語

```
int GpibExecDeviceClear (
  ULONG   BoardNo,
  PLONG   AdrsTbl
);
```

●Visual Basic

```
Declare Function GpibExecDeviceClear Lib "gpc43042.dll"( _
  ByVal   BoardNo  As Long,        _
  ByRef   AdrsTbl  As String       _
) As Long
```

●Delphi

```
function GpibExecDeviceClear (
     BoardNo  : Cardinal;
     AdrsTbl  : Pointer
) : Integer; stdcall; external 'GPC43042.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

AdrsTbl

機器のアドレステーブルを指定します。

デバイスクリアを有効にする機器のアドレステーブルへのポインタ（参照渡し）を指定します。機器アドレステーブルは次の形式で指定します。

1 台の機器（アドレス 10）の場合
LONG 配列
+0[10]
+1[-1]…終端コード

2 台の機器（アドレス 10, 11）の場合
LONG 配列
+0[10]
+1[11]
+2[-1]…終端コード

※-1（終端コード）は必ず最後に付加してください。

【戻り値】

正常終了した場合は、0 が返されます。

0 以外の値が返された場合については、『4.4 戻り値一覧』をご参照ください。

【備考】

本関数を実行することにより、計測機器に対してデバイスクリアを実行します。デバイスクリアを受信した計測機器の動作については、計測機器のマニュアルをご参照ください。

【使用例】

●C 言語

```
int Ret;
ULONG BoardNo;
LONG AdrTbl[] = { 10, -1 };  // 機器アドレステーブル
BoardNo = 0;
Ret = GpibExecDeviceClear( BoardNo, &AdrTbl[0] );
```

●Visual Basic

```
Dim Ret As Long
Dim BoardNo As Long
Dim AdrTbl(1) As Long
BoardNo = 0
AdrTbl(0) = 10
AdrTbl(1) = -1
Ret = GpibExecDeviceClear( BoardNo, AdrTbl(0) )
```

●Delphi

```
var
  Ret : Integer;
  BoardNo : Cardinal;
  AdrTbl : Array[0..1] of Integer;

BoardNo := 0;
AdrTbl[0] := 10;
AdrTbl[1] := -1;
Ret := GpibExecDeviceClear( BoardNo, AdrTbl[0] );
```

ボード番号0のGP-IBインタフェースモジュールからGP-IBアドレス10の機器に対して、デバイスクリアを実行します。

7. GpibCheckSrq

【機能】

機器から SRQ 信号を受け付けているかどうか確認します。

【書式】

●C 言語

```
int GpibCheckSrq (
  ULONG      BoardNo
);
```

●Visual Basic

```
Declare Function GpibCheckSrq Lib "gpc43042.dll"( _
  ByVal   BoardNo    As Long    _
) As Long
```

●Delphi

```
function GpibCheckSrq (
      BoardNo   : Cardinal
) : Integer; stdcall; external 'GPC43042.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

【戻り値】

正常終了した場合は、以下の値が返されます。

戻り値	意味
5	SRQ信号を受け付けていません。
4	SRQ信号を受け付けています。

上記以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【使用例】

●C 言語

```
int Ret;
ULONG BoardNo;
BoardNo = 0;
Ret = GpibCheckSrq( BoardNo );
```

●Visual Basic

```
Dim Ret As Long
Dim BoardNo As Long
BoardNo = 0
Ret = GpibCheckSrq( BoardNo )
```

●Delphi

```
var
  Ret : Integer;
  BoardNo : Cardinal;

BoardNo := 0;
Ret := GpibCheckSrq( BoardNo );
```

機器からSRQ信号を受け付けているかどうか確認します。

8. GpibExecSpoll

【機能】

シリアルポーリングを実行します。

【書式】

●C 言語

```
int GpibExecSpoll (
    ULONG    BoardNo,
    PLONG    AdrsTbl,
    PLONG    StbTbl,
    PLONG    StbAdrs
);
```

●Visual Basic

```
Declare Function GpibExecSpoll Lib "gpc43042.dll"( _
    ByVal    BoardNo      As Long,    _
    ByRef    AdrsTbl      As Long,    _
    ByRef    StbTbl       As Long,    _
    ByRef    StbAdrs      As Long     _
) As Long
```

●Delphi

```
function GpibExecSpoll (
        BoardNo    : Cardinal;
        AdrsTbl    : Pointer;
        StbTbl     : Pointer;
        StbAdrs    : Pointer
) : Integer; stdcall; external 'GPC43042.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

AdrsTbl

シリアル・ポールを行う機器のアドレステーブルへのポインタ（参照渡し）を指定します。機器アドレステーブルは次の形式で指定します。

1 台の機器（アドレス 10）の場合
LONG 配列
+0[10]
+1[-1]…終端コード

2 台の機器（アドレス 10, 11）の場合
LONG 配列
+0[10]
+1[11]
+2[-1]…終端コード

※-1（終端コード）は必ず最後に付加してください。

StbTbl

シリアル・ポーリングを行い受信した各機器の有効なステータス・バイトを格納するテーブルへのポインタ（参照渡し）を指定します。
本関数呼び出し後、終端に-1 が付加されます。
※AdrsTbl（シリアルポーリング機器アドレステーブル）と同じサイズの領域を確保して呼び出してください。

StbAdrs

SRQ を送出し、有効なステータス・バイトを持っていた各機器のアドレスを格納するテーブルへのポインタ（参照渡し）を指定します。
本関数呼び出し後、終端に-1 が付加されます。
※AdrsTbl（シリアルポーリング機器アドレステーブル）と同じサイズの領域を確保して呼び出してください。

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

AdrsTbl(シリアルポーリング機器アドレステーブル)、StbTbl(受信 STB 格納テーブル)、StbAdrs(STB 送信機器アドレステーブル)の各ポインタ（参照渡し）には次のような手順でデータを格納、取得します。

1）関数呼び出し前、呼び出し時

シリアルポーリングを行う機器のアドレスを AdrsTbl(シリアルポーリング機器アドレステーブル)に順次格納し、終端-1 を設定しておきます。
StbTbl と StbAdrs は、関数からのリターン時にそれぞれデータが格納されて戻ってくるため、必ず AdrsTbl と同じサイズの領域を確保して関数に指定します。そのとき、確保領域には何もデータを入れておく必要はありません(領域の確保さえしておけば問題はありません)。

2）関数の動作、リターン時

AdrsTbl に格納されている機器に対して順次シリアルポーリングを行っていきます。
このとき、機器からの有効な STB（ステータスバイト）が受信できた場合には、その STB を StbTbl ポインタの領域へ格納します。また、そのときの機器アドレスを StbAdrs ポインタに格納します(ポインタは当然格納後に更新されます)。
最後までシリアルポーリングを行った時点で、StbTbl ポインタと StbAdrs ポインタの指す領域に終端コード'-1'が設定されます。
GP-IB アドレス 10、11 の機器に対してシリアルポーリングを行い、アドレス 11 の機器が有効な STB を送信した場合には以下のようなコードと実行結果になります。

3) <コード>

```
ULONG BoardNo;
LONG AdrsTbl[3], StbTbl[3], StbAdrsTbl[3];
int Ret, i;

BoardNo = 0;
AdrsTbl[0] = 10; // 1台目
AdrsTbl[1] = 11; // 2台目
AdrsTbl[2] = -1; // 終端コード
Ret = GpibExecSpoll(BoardNo, &AdrsTbl[0], &StbTbl[0], &StbAdrsTbl[0]);
If (Ret == 0) {          // 正常終了
  for (i = 0; StbTbl[i] != -1; i++) {
     printf("STB 送出機器アドレス : %d, 有効 STB 値 : %x¥n", StbTbl[i], StbAdrsTbl[i]);
  }
} else {                 // 異常終了
  …
}
```

<実行結果>
STB 送出機器アドレス : 11, 有効 STB 値 : 41

【使用例】

●C 言語

```
int Ret;
ULONG BoardNo;
LONG AdrsTbl[3], StbTbl[3], StbAdrsTbl[3];
BoardNo = 0;
AdrsTbl[0] = 10; // 1台目
AdrsTbl[1] = 11; // 2台目
AdrsTbl[2] = -1; // 終端コード
Ret = GpibExecSpoll( BoardNo, &AdrsTbl[0], &StbTbl[0], &StbAdrsTbl[0] );
```

●Visual Basic

```
Dim Ret As Long
Dim BoardNo As Long
Dim AdrTbl(2) As Long
Dim StbTbl(2) As Long
Dim StbAdr(2) As Long
nBoardNo = 0
AdrTbl(0) = 10
AdrTbl(1) = 11
AdrTbl(2) = -1
Ret = GpibExecSpoll( BoardNo, AdrTbl(0), StbTbl(0), StbAdr(0) )
```

●Delphi

```
var
  Ret : Integer;
  BoardNo : Cardinal;
  AdrTbl : Array[0..2] of Integer;
  StbTbl : Array[0..2] of Integer;
  StbAdr : Array[0..2] of Integer;

BoardNo := 0;
AdrTbl[0] := 10;
AdrTbl[1] := 11;
AdrTbl[2] := -1;
Ret := GpibExecSpoll( BoardNo, @AdrTbl[0], @StbTbl[0], @StbAdr[0] );
```

ボード番号 0 の GP-IB インタフェースモジュールから GP-IB アドレス 10、11 の計測機器へ、シリアルポーリングを実行します。

9. GpibReceive

【機能】

計測機器から受信を行います。

【書式】

●C 言語

```
int GpibReceive (
  ULONG    BoardNo,
  PLONG    AdrsTbl,
  PULONG   Length,
  PVOID    Buffer
);
```

●Visual Basic

バイト型/数値型変数のデータ受信の場合

```
Declare Function GpibReceive Lib "gpc43042.dll"( _
  ByVal  BoardNo   As Long,        _
  ByRef  AdrsTbl   As Long,        _
  ByRef  Length    As Long,        _
  ByRef  Buffer    As Any          _
) As Long
```

文字列型変数のデータ受信の場合

```
Declare Function GpibReceive Lib "gpc43042.dll"( _
  ByVal  BoardNo   As Long,        _
  ByRef  AdrsTbl   As Long,        _
  ByRef  Length    As Long,        _
  ByVal  Buffer    As String       _
) As Long
```

●Delphi

```
function GpibReceive (
       BoardNo  : Cardinal;
       AdrsTbl  : Pointer;
  var  Length   : Cardinal;
       Buffer   : Pointer
) : Integer; stdcall; external 'GPC43042.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

AdrsTbl

機器のアドレステーブルへのポインタ（参照渡し）を指定します（『3.6　機器アドレス

テーブルについて』参照)。
アドレステーブルへの終端には必ず-1 を設定します。

Length

受信バッファのバイトサイズが格納されている変数へのポインタ（参照渡し）を指定します。本関数呼び出し後、実際に受信したバイトサイズが格納されます。
受信バッファの領域は、必ず「受信データ長」＋「デリミタ」分の領域を確保するようにしてください（例えば、受信データ長が 16 バイト、デリミタが「CRLF+EOI」の場合には、16+2 の 18 バイト以上の領域を確保する必要となります)。

Buffer

受信バッファ領域へのポインタ（参照渡し）を指定します。

※ Visual Basic の場合には、API の Declare Function の宣言がバイト型/数値型と文字列型とでは異なります。
※ Visual Basic では、文字列変数として固定長文字列を指定してください。
例: Dim RecvBuffer As String * 32 など

【戻り値】

正常終了しますと、以下の値を返します。

戻り値	意味
2	EOIを検出して受信が完了しました。
1	指定された受信データ数に達して受信が完了しました。
0	デリミタを検出して受信が完了しました。

上記以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

・ 機器からの応答がタイムアウト時間を経過しても無い場合には、タイムアウトエラーとして転送中断が行われます。

【使用例】

●C 言語

```
int Ret;
ULONG BoardNo;
LONG AdrsTbl[] = { 10, -1 };  // 機器アドレステーブル
Static ULONG Length;
Static BYTE RecvBuff[80];
BoardNo = 0;
Length = 80;
Ret = GpibReceive( BoardNo, &AdrsTbl[0], &Length, &RecvBuff[0] );
```

●Visual Basic

```
Dim Ret As Long
Dim BoardNo As Long
Dim AdrTbl(1) As Long
Public Dim Length As Long
Public Dim RecvBuff(80) As Byte
BoardNo = 0
Length = 80
AdrTbl(0) = 10
AdrTbl(1) = -1
Ret = GpibReceive( BoardNo, AdrTbl(0), Length, RecvBuff(0) )
```

●Delphi

```
var
  Ret : Integer;
  BoardNo : Integer;
  AdrTbl : Array[0..1] of Integer;
  Length : Cardinal;
  RecvBuff : Array[0..79] of Char;

BoardNo := 0;
Length := 80;
AdrTbl[0] := 10;
AdrTbl[1] := -1;
Ret := GpibReceive( BoardNo, @AdrTbl[0], Length, @RecvBuff[0] );
```

ボード番号 0 の GP-IB インタフェースモジュールへ GP-IB アドレス 10 の機器から 80 バイトのデータを読み込みます。

10. GpibSend

【機能】

計測機器へ送信を行います。

【書式】

●C 言語

```
int GpibSend (
  ULONG    BoardNo,
  PLONG    AdrsTbl,
  ULONG    Length,
  PVOID    Buffer
);
```

●Visual Basic

バイト型/数値型変数のデータ送信の場合

```
Declare Function GpibSend Lib "gpc43042.dll"( _
  ByVal  BoardNo  As Long,        _
  ByRef  AdrsTbl  As Long,        _
  ByVal  Length   As Long,        _
  ByRef  Buffer   As Any          _
) As Long
```

文字列型変数のデータ送信の場合

```
Declare Function GpibSend Lib "gpc43042.dll"( _
  ByVal  BoardNo  As Long,        _
  ByRef  AdrsTbl  As Long,        _
  ByVal  Length   As Long,        _
  ByVal  Buffer   As String       _
) As Long
```

●Delphi

```
function GpibSend (
      BoardNo  : Cardinal;
      AdrsTbl  : Pointer;
      Length   : Cardinal;
      Buffer   : Pointer
) : Integer; stdcall; external 'GPC43042.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

AdrsTbl

機器のアドレステーブルへのポインタ（参照渡し）を指定します（『3.6　機器アドレス

テーブルについて』参照)。
アドレステーブルへの終端には必ず-1 を設定します。

Length
送信データ長のバイトサイズを指定します。
この送信データ長+デリミタ分のサイズが GP-IB バスへ送信されます。
デリミタは自動で付加されます。
従って、送信データ長が 10 バイトで、デリミタが「CRLF+EOI」の場合には、計 12 バイトのデータが送信されます。

Buffer
送信バッファ領域へのポインタ（参照渡し）を指定します。
※ Visual Basic の場合には、API の Declare Function の宣言がバイト型/数値型と文字列型とでは異なります。

【戻り値】
正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】
・ 機器からの応答がタイムアウト時間を経過しても無い場合には、タイムアウトエラーとして転送中断が行われます。

【使用例】
●C 言語
```
int Ret;
ULONG BoardNo;
LONG AdrsTbl[] = { 10, -1 };  // 機器アドレステーブル
Static ULONG Length;
Static PBYTE SendData = "TEST";
BoardNo = 0;
Length = 4;
Ret = GpibSend( BoardNo, &AdrsTbl[0], Length, SendData );
```

●Visual Basic

```
Dim Ret As Long
Dim BoardNo As Long
Dim AdrTbl(1) As Long
Dim Length As Long
Dim SendData As String
BoardNo = 0
lLength = 4
AdrTbl(0) = 10
AdrTbl(1) = -1
SendData = "TEST"
Ret = GpibSend( BoardNo, AdrTbl(0), Length, SendData(0) )
```

●Delphi

```
var
  Ret : Integer;
  BoardNo : Integer;
  AdrTbl : Array[0..1] of Integer;
  Length : Cardinal;
  SendData : Array[0..3] of Char;

BoardNo := 0;
lLength := 4;
AdrTbl[0] := 10;
AdrTbl[1] := -1;
SendData[0] := 'T';
SendData[1] := 'E';
SendData[2] := 'S';
SendData[3] := 'T';
Ret := GpibSend( BoardNo, @AdrTbl[0], Length, @SendData[0] );
```

ボード番号 0 の GP-IB インタフェースモジュールから GP-IB アドレス 10 の機器へ 4 バイトのデータ（TEST）を送信します。

11. GpibClose

【機能】

インタフェースモジュールをクローズします。

【書式】

●C 言語

```
int GpibClose (
  ULONG   BoardNo
);
```

●Visual Basic

```
Declare Function GpibClose Lib "gpc43042.dll"( _
  ByVal   BoardNo    As Long     _
) As Long
```

●Delphi

```
function GpibClose (
     BoardNo  : Cardinal
) : Integer; stdcall; external 'GPC43042.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】

・ 再度、インタフェースモジュールへのアクセスを行う場合にはオープン処理(GpibOpen 関数)を呼び出してください。

【使用例】

●C 言語

```
int Ret;
ULONG BoardNo;
BoardNo = 0;
Ret = GpibClose( BoardNo );
```

●Visual Basic

```
Dim Ret As Long
Dim BoardNo As Long
BoardNo = 0
Ret = GpibClose( BoardNo )
```

●Delphi

```
var
  Ret : Integer;
  BoardNo : Cardinal;

BoardNo := 0;
Ret := GpibClose( BoardNo );
```

インタフェースモジュールをクローズします。

12. GpibSetSrqEvent

【機能】

SRQ コールバックイベントを登録します。
ただし、Visual Basic Ver4.0 では、コールバック関数の登録ができない為、イベント機能を使用することはできません。

【書式】

●C 言語(x86)

```
int GpibSeqSrqEvent (
    ULONG           BoardNo,
    LPSRQCALLBACK   SrqProc,
    DWORD           dwUser
);
```

●C 言語(x64)

```
int GpibSeqSrqEvent (
    ULONG           BoardNo,
    LPSRQCALLBACK   SrqProc,
    PVOID           dwUser
);
```

●Visual Basic

```
Declare Function GpibSeqSrqEvent Lib "gpc43042.dll"( _
    ByVal   BoardNo     As Long,    _
    ByVal   SrqProc     As Long,    _
    ByVal   dwUser      As Long     _
) As Long
```

●Delphi

```
function GpibSeqSrqEvent (
      BoardNo    : Cardinal;
      SrqProc    : FARPROC;
      dwUser     : DWORD
) : Integer; stdcall; external 'GPC43042.DLL';
```

【パラメータ】

BoardNo
　ボードアクセス番号(0～15)を指定します。

SrqProc
　コールバック関数のアドレスを指定します。
　『4.3　コールバック関数』をご参照ください。

dwUser
　コールバック関数に渡す任意のデータを指定します。

【戻り値】
正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【備考】
コールバック関数の書式は以下の通りです（C言語(x86))。『4.3　コールバック関数』もご参照ください。
●C 言語(x86)

```
void CALLBACK EventProc (
    int       BoardNo,
    DWORD     dwUser
);
```

【使用例】
●C 言語(x86)

```
void CALLBACK OnSrqProc (int BoardNo,DWORD dwUser) {
       … // イベントに対応する処理を記述します
}

ULONG BoardNo;

Ret = GpibSetSrqEvent( BoardNo, (PLPSRQCALLBACK)OnSrqProc, 0 );
```

●C 言語(x64)

```
void CALLBACK OnSrqProc (int BoardNo,PVOID dwUser) {
       … // イベントに対応する処理を記述します
}

ULONG BoardNo;

Ret = GpibSetSrqEvent( BoardNo, (PLPSRQCALLBACK)OnSrqProc, 0 );
```

●Visual Basic
コールバックルーチンはGpibSetSrqEvent関数の呼び出しを行うプロジェクト内の標準モジュールの中に記述しなければなりません。

GpibSetSrqEvent 関数の引数パラメータでプロシージャのアドレスを渡す為に AddressOf 演算子を使用します。(GpibSetSrqEvent関数の使用例を参照してください。)
AddressOf 演算子を使うと、プロシージャからの戻り値ではなく、プロシージャ自体のアドレスが､ダイナミック リンク ライブラリ (DLL)のGpibSetSrqEvent関数に渡されます。

```
Sub OnSrqProc(BoardNo As Long, dwUser As Long)
        … 'イベントに対応する処理を記述します。
End Sub

Dim Ret As Long
Dim BoardNo As Long
Dim dwUser As Long
Ret = GpibSetSrqEvent( BoardNo, AddressOf OnSrqProc, dwUser)
```

●Delphi

```
procedure OnSrqProc(BoardNo : DWORD, dwUser : DWORD); stdcall;
var
      //変数定義

begin
      … //イベントに対応する処理を記述します。
end;

var
  Ret : Integer;
  BoardNo : Cardinal;
  dwUser : DWORD;

Ret := GpibSetSrqEvent( BoardNo, OnSrqProc, dwUser );
```

SRQコールバック関数を登録します。

13. GpibWaitSrqEvent

【機能】

SRQ コールバックイベントを待ちます。

【書式】

●C 言語

```
int GpibWaitSrqEvent (
    ULONG    BoardNo,
    ULONG    Timeout
);
```

●Visual Basic

```
Declare Function GpibWaitSrqEvent Lib "gpc43042.dll"( _
    ByVal    BoardNo    As Long,    _
    ByVal    Timeout    As Long     _
) As Long
```

●Delphi

```
function GpibWaitSrqEvent (
      BoardNo    : Cardinal;
      Timeout    : Cardinal
) : Integer; stdcall; external 'GPC43042.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

Timeout

タイムアウト時間をミリ秒単位で指定します。

タイムアウト時間が経過すると、関数はオブジェクトの状態が非シグナル状態である場合でも、制御を戻します。0 を指定した場合は、状態を調べてからすぐに制御を戻します

【戻り値】

正常終了した場合は、0 が返されます。

0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【使用例】

●C 言語

```
int Ret;
ULONG BoardNo;
BoardNo = 0;
Ret = GpibWaitSrqEvent( BoardNo, 5000 );
```

●Visual Basic

```
Dim Ret As Long
Dim BoardNo As Long
BoardNo = 0
Ret = GpibWaitSrqEvent( BoardNo, 5000 )
```

●Delphi

```
var
  Ret : Integer;
  BoardNo : Cardinal;

BoardNo := 0;
Ret := GpibWaitSrqEvent( BoardNo, 5000 );
```

SRQ コールバックイベント待ちをします。(タイムアウト 5 秒)

14. GpibKillSrqEvent

【機能】

SRQ コールバックイベントの登録を解除します。

【書式】

●C 言語

```
int GpibKillSrqEvent (
   ULONG   BoardNo
);
```

●Visual Basic

```
Declare Function GpibKillSrqEvent Lib "gpc43042.dll"( _
   ByVal   BoardNo   As Long   _
) As Long
```

●Delphi

```
function GpibKillSrqEvent (
     BoardNo  : Cardinal
) : Integer; stdcall; external 'GPC43042.DLL';
```

【パラメータ】

BoardNo

ボードアクセス番号(0～15)を指定します。

【戻り値】

正常終了した場合は、0 が返されます。
0 以外の値が返された場合については、『4.4　戻り値一覧』をご参照ください。

【使用例】

●C 言語

```
int Ret;
ULONG BoardNo;
BoardNo = 0;
Ret = GpibKillSrqEvent( BoardNo );
```

●Visual Basic

```
Dim Ret As Long
Dim BoardNo As Long
BoardNo = 0
Ret = GpibKillSrqEvent( BoardNo )
```

●Delphi

```
var
  Ret : Integer;
  BoardNo : Cardinal;

BoardNo := 0;
Ret := GpibKillSrqEvent( BoardNo );
```

SRQ コールバックイベント登録を解除します。

4.3 コールバック関数

※Visual Basicでコールバック機能を使用する場合、幾つかの制約事項が発生します(後述の『4.3.2 Visual Basic使用時の制約事項』をご参照下さい)。
Visual Basic で 非同期のイベント処理を行うには、弊社Web siteより無償で提供しております。
BPA- 0515「GP-IB用 ActiveXコントロール」 をご使用になることを推奨致します。

4.3.1 PciGpibExSetSrqEvent関数、およびGpibSetSrqEvent関数により設定されるコールバック関数

PciGpibExSetSrqEvent関数、およびGpibSetSrqEvent関数にて登録し、SRQイベントが発生した時にコールされるコールバックルーチンです。

【記述】

コールバックルーチンを使用する場合、下記のように記述します。
（下記はコールバックルーチンを OnSrqEvent とする場合の例です。）

●C 言語(x86)

```
void CALLBACK OnSrqEvent(int BoardNo, DWORD dwUser)
{
   // 割り込みイベントに対応する処理を記述します
}
```

コールバックルーチンの関数型 LPSRQCALLBACK は下記のように定義されます。

```
typedef void (CALLBACK LPSRQCALLBACK)(DWORD EventMask, DWORD dwUser);
typedef SRQCALLBACK FAR *LPSRQCALLBACK;
```

●C 言語(x64)

```
void CALLBACK OnSrqEvent(int BoardNo, PVOID dwUser)
{
   // 割り込みイベントに対応する処理を記述します
}
```

コールバックルーチンの関数型 LPSRQCALLBACK は下記のように定義されます。

```
typedef void (CALLBACK LPSRQCALLBACK)(DWORD EventMask, PVOID dwUser);
typedef SRQCALLBACK FAR *LPSRQCALLBACK;
```

●Visual Basic

```
Function OnSrqEvent(ByVal BoardNo As Long, ByVal dwUser As Long)
    ‘ 割り込みイベントに対応する処理を記述します
End Sub
```

コールバックルーチンはPciGpibExSetSrqEvent関数、およびGpibSetSrqEvent関数の呼び出しを行うプロジェクト内の標準モジュールの中に記述しなければなりません。
PciGpibExSetSrqEvent関数、およびGpibSetSrqEvent関数の引数パラメータでプロシージャのアドレスを渡す為にAddressOf演算子を使用します。(PciGpibExSetSrqEvent関数、およびGpibSetSrqEvent関数の使用例を参照してください。)
AddressOf演算子を使うと、プロシージャからの戻り値ではなく、プロシージャ自体のアドレスが、ダイナミック リンク ライブラリ (DLL)のPciGpibExSetSrqEvent関数、およびGpibSetSrqEvent関数に渡されます。

●Delphi

```
procedure OnSrqEvent(BoardNo:Integer; dwUser:DWORD); stdcall;
begin
    // 割り込みイベントに対応する処理を記述します
end;
```

【パラメータ】

BoardNo

イベントが発生したボード番号が格納されます。

dwUser

ユーザデータが格納されます。エラーが発生した場合、このパラメータは0になります。

【戻り値】

コールバックルーチンに戻り値はありません。

4.3.2 Visual Basic使用時の制約事項

●Visual Basic Ver. 4.0では、コールバック関数の登録ができない(AddressOf演算子がサポートされない)為、イベント機能を使用することはできません。Visual Basic Ver. 5.0以降は可能です。

Microsoft Visual Basic 6.0上で弊社ソフトウェアライブラリが提供する関数コールバック機能を使用した場合、下記のアプリケーションエラーが発生する場合があります。

「"0x660d64d0"の命令が"0x0000009c"のメモリを参照しました。メモリが"written"になることはできませんでした。」
※"0x660d64d0"は異なる場合があります

アプリケーションエラーは下記の条件で発生します。
○登録したコールバック関数内で、下記の関数、ステートメントをコールする。
・関数コール(弊社ソフトウェアライブラリが提供する関数含む)
・Visual Basicのステートメント(Str()など)
・スタティックテキストへの文字列代入
など

また、この問題は、Visual Basic 6.0のLearning、Professional、Enterprise Edition

のすべてに当てはまり、サービスパックの適用有無にかかわらず発生します。

コールバック関数内で、アプリケーションエラーを発生させる処理を行わずに別の機能を利用することで目的の処理が実行できるように設計を変更してください。

例えば、コールバック関数の登録以外の通知機能、メッセージ、イベントオブジェクトが提供されている場合は、その機能を利用して下さい。また、割り込みなどの必要な情報取得が、関数（ステータスを取得する関数など）の呼び出しで可能な場合は、定期的に要因をチェックするポーリング処理に変更して下さい。
弊社としては、関数コールバックを用いた非同期、割り込み処理を、Visual Basic 6.0 上で 構築される場合、これらの問題を回避する為に、GP-IB ActiveX コントロールの使用をお奨め します。
(GP-IB ActiveX コントロールは、弊社 Web site から無料ダウンロード出来る BPA-0515 に添付されております)

Microsoft Visual Basic 6.0 は、スレッディング モデルとして、アパートメント モデルを採用しています。
Microsoft Visual Basic 6.0 が作成、起動したスレッド以外からコールバック関数が実行された場合にアプリケーションエラーが発生する場合があります。

弊社ソフトウェアライブラリでは、ライブラリ内で起動した別のスレッドから登録されたコールバック関数の実行を行いますので、この問題が発生します。

解決方法は、以下の項を参照してください。

●Microsoft Visual Basic で非同期のイベント処理を行う方法について
Visual Basic で 非同期のイベント処理を行うには、BPA-0515「GP-IB用 ActiveXコントロール」を御使用下さい。
BPA-0515「GP-IB用 ActiveXコントロール」 を使用する事で、イベント等の各種処理が行えます。BPA- 0515「GP-IB用 ActiveXコントロール」 は 弊社Web site よりダウンロードしてご使用ください。

4.4 戻り値一覧

エラー識別子	値	意味
GPIB_SUCCESS_NOT_FOUND_LISTENER	11	リスナが 1台も見つかりませんでした。
GPIB_SUCCESS_OK_SEND_STB	7	ステータス・バイトはコントローラへ通知済です。
GPIB_SUCCESS_NOT_EXEC_SPOLL	6	まだ、シリアル・ポールは行われていません。
GPIB_SUCCESS_NOT_ACTIVE_SRQ	5	SRQ信号無効です。 PciGpibExCheckSrq関数において、SRQ割り込みを受け付けていないとき、この戻り値が通知されます。
GPIB_SUCCESS_ACTIVE_SRQ	4	SRQ信号有効です。 PciGpibExCheckSrq関数において、SRQ割り込みを受け付けているとき、この戻り値が通知されます。
GPIB_SUCCESS_OK_EOI_DETECT	2	EOIを検出して終了しました。 データ受信時にデリミタとしてEOIを検出して終了しました。
GPIB_SUCCESS_OK_RECV_DATA_CNT	1	指定された受信データ数に達して終了しました。 データ受信時に指定された受信データ数に達して終了しました。この状態ではデリミタの検出は行われていません。デリミタ有りの送受信処理を行っている場合には、引き続いてデータ受信処理を実行する必要があります。
GPIB_SUCCESS_OK_DELIM_DETECT	0	データ受信時に指定された受信デリミタを検出して終了しました。
GPIB_SUCCESS	0	処理を正常終了しました。
GPIB_ERROR_ILLEGAL_BOARDNO	-1	ボードアクセス番号が違います。 ・コントロールパネルで指定されたボードアクセス番号以外のインタフェースモジュールを指定しました。 ・すでに初期化済のインタフェースモジュールに対して再初期化を行おうとしました。 ・または初期化が行われていないインタフェースモジュールに対して制御を行おうとしました。 ・0～15以外の番号を指定しました。
GPIB_ERROR_ILLEGAL_INP_PARAM	-2	入力パラメータに間違いがあります。 入力パラメータの値、範囲を確認してください。
GPIB_ERROR_ILLEGAL_PARAM_NO	-3	パラメータ番号に間違いがあります。 パラメータ番号の値、範囲を確認してください。
GPIB_ERROR_NOT_USE_SLAVE	-4	非コントローラ状態では使用できません。 非コントローラモードにおいて、コントローラモードでのみ使用可能な関数を呼び出しました。 (インタフェースモジュールが[CIC]状態で無い場合に、[CIC]状態でのみ使用可能なAPIを呼び出しました)
GPIB_ERROR_NOT_USE_MASTER	-5	コントローラ状態では使用できません。 コントローラモードにおいて、非コントローラモードでのみ使用可能な関数を呼び出しました。 (インタフェースモジュールが[CIC]状態の場合に、非[CIC]状態でのみ使用可能なAPIを呼び出しました)
GPIB_ERROR_NOT_BOTH_USE	-6	コントローラ/非コントローラの両方の指定はできません。 PciGpibExSetSignal関数において、コントローラインチャージのみと非コントローラインチャージのみの両方の事象変化検出設定を行おうとしました。

GPIB_ERROR_CMD_TMO	-7	バスコマンドの送出に失敗しました。 ・コントローラ・イン・チャージにおいて、バスコマンドの送出に失敗しました。以下の要因が考えられます。 ・システムコントローラにおいて、PciGpibExSetIfc関数を実行していない。 ・現在、他の機器および非同期入出力中により、GP-IBバスが占有状態である場合 ・非コントローラ・イン・チャージにおいて、バスコマンドを送出する関数を実行した場合 ・他のアプリケーションの負荷が重く、バスコマンド送出タイムアウトが発生した場合(この場合、コントロールパネルでタイムアウト時間を増やすようにしてください。)
GPIB_ERROR_NO_SET_SIGNAL	-8	検出する事象が設定されていません。 PciGpibExSetSignal関数で事象変化検出設定を行わずに、PciGpibExWaitSignal関数を実行しようとしました。
GPIB_ERROR_ACTIVE_SRQ	-9	シリアル・ポーリングにも関わらずSRQ送出元を検出できませんでした。
GPIB_ERROR_STB_TMO	-10	STB受信時にタイムアウトが発生しました。 ・機器からのステータス・バイトが指定した時間内に受信できませんでした。 ・コントロールパネルの設定「STB応答時間」を増やす必要があります。
GPIB_ERROR_DATA_RECV	-12	データ受信に失敗しました。 ・データ受信処理に失敗しました。以下の要因が考えられます。 ・他の機器もしくは非同期転送中により、GP-IBバスが占有中である場合
GPIB_ERROR_DATA_SEND	-13	データ送信に失敗しました。 ・データ送信処理に失敗しました。以下の要因が考えられます。 ・送信したデータが受信されなかった(この場合、ケーブル接続、全ての機器の電源が入っているか、ケーブル長はGP-IBの規格を守っているか等を確認してください。 ・他の機器もしくは非同期転送中により、GP-IBバスが占有中である場合
GPIB_ERROR_TRANS_TMO	-14	タイムアウトが発生しました。 ・送受信時に指定時間内にデータ転送が終了しませんでした。以下の要因が考えられます。 ・指定したデータ長に対するデータ転送時間がタイムアウト時間より長い場合(この場合は送受信タイムアウト時間を長くする必要があります。) ・送受信中に機器からの応答が何らかの要因でなくなった場合
GPIB_ERROR_WAIT_SIGNAL_TMO	-15	タイムアウトで終了しました。 PciGpibExWaitSignal関数において、指定時間内に事象変化が検出できませんでした。
GPIB_ERROR_IFC_TRANS_EXIT	-16	IFC受信による強制終了。 送受信中にIFCを受信したため、送受信処理を強制終了しました。
GPIB_ERROR_NOT_CACS	-19	コントローラアクティブ状態に遷移できませんでした。
GPIB_ERROR_NOW_BUS_OCCUPATION	-20	現在、バスが占有状態となっています(非同期入出力中)。 ・送受信関数を実行しましたが、GP-IBバスが占有状態となっています。 ・非同期入出力の完了を待つようにしてください。
GPIB_ERROR_NO_SET	-22	設定変更ができませんでした。 ・PciGpibExSetConfig関数において、設定変更に失敗しました。以下の要因が考えられます。 ・非同期入出力中にPciGpibExSetConfig関数を呼び出しました。
GPIB_ERROR_NOT_START_TIMER	-24	タイマがスタートできませんでした。
GPIB_ERROR_EXIST_START_TIMER	-25	すでにタイマはスタートしています。
GPIB_ERROR_NOT_TICK_TIMER	-26	タイマが動作中ではありません。

GPIB_ERROR_FILE_ACCESS	-30	ファイルがオープンできません、またはファイル読み込み/書き込み中にエラーが発生しました。 以下の要因が考えられます。 ・存在しないファイルをオープンしようとしました。 ・ディスクの空き容量が足らない。 ・空きメモリ容量が極端に少ない(ハードディスクの空き容量に注意してください。) ・共有ファイルをオープンしようとした、またはそのファイルが別のプロセスでオープンされている。
GPIB_ERROR_SET_CALLBACK_EVENT	-40	コールバックイベントの登録に失敗しました。 既に同じボードアクセス番号にてPciGpibExSetSrqEvent関数にて登録済です。
GPIB_ERROR_KILL_CALLBACK_EVENT	-41	コールバックイベントの登録解除に失敗しました。
GPIB_ERROR_WAIT_OBJECT_SIGNAL	-42	イベントオブジェクトが有効です。 PciGpibExWaitSrqEvent関数実行前にイベントが有効済となっています。
GPIB_ERROR_WAIT_OBJECT_TMO	-43	イベントオブジェクト待ちにてタイムアウトが発生しました。 PciGpibExWaitSrqEvent関数実行にてタイムアウト時間内にイベントが有効になりませんでした。
GPIB_ERROR_WAIT_OBJECT_FAILED	-44	イベントオブジェクト待ちにてエラーが発生しました。 ・PciGpibExWaitSrqEvent関数実行にてイベント待ちにエラーが発生しました。 ・イベントリソースが足りなくなっている可能性があります。 ・起動中のアプリケーションの数を少なくしてください。
GPIB_ERROR_NOT_SET_CALLBACK_EVENT	-45	まだコールバックイベントが登録されていません。 PciGpibExSetSrqEvent関数を実行してコールバックイベントの登録を行ってください。
GPIB_ERROR_NOT_SYS_CONTROLLER	-50	システムコントローラではありません。
GPIB_ERROR_NOT_CREATE_EVENT	-992	イベントの作成ができませんでした。 以下の要因が考えられます。 ・システムリソースが極端に不足している可能性があります。
GPIB_ERROR_NOT_CREATE_THREAD	-993	スレッドの作成ができませんでした。 以下の要因が考えられます。 ・システムリソースが極端に不足している可能性があります。
GPIB_ERROR_NOT_INIT_CALL	-994	インタフェースモジュールの初期化が行われていない状態では、使用できません。
GPIB_ERROR_NOT_ALLOC_PAGE	-995	インタフェースモジュールの使用終了ができませんでした。 以下の要因が考えられます。 ・割り込みの制御に失敗しました(このエラーが発生した場合、Windowsのシステムに問題があります。リソースの競合がないか確認してください。)。
GPIB_ERROR_NOT_BOARD	-996	ドライバ側のページの確保ができませんでした。 以下の要因が考えられます。 ・空きメモリが極端に少ない。 ・ハードディスクの空き容量が極端に少ない場合(1MB以下等) ・他に動作しているプロセスが多く存在する。 ・本エラーが発生する場合は、一度に要求するデータサイズを少なくしていくか、もしくはメモリの増設をお勧めします。
GPIB_ERROR_NOT_USE_TIMER	-997	タイマ設定に失敗しました。 以下の要因が考えられます。 ・他のプロセスにおいて、タイマを多く使用している場合。本エラーが発生する場合は、他に動作しているプロセスを終了させるようにしてください。

GPIB_ERROR_NOT_USE_RESOURCE	-998	割り込みが使用できません。 以下の要因が考えられます。 ・割り込みのリソースが競合している、もしくは使用中となっている。 ・リソースが競合しないようにしてください。
GPIB_ERROR_NOT_BOARD	-999	インタフェースモジュールが存在しません。 以下の要因が考えられます。 ・インタフェースモジュール自体に異常がある場合 ・自己診断プログラムを実行して、インタフェースモジュールのチェックを行ってください。
GPIB_ERROR_USBIO_FAILED	-4000	USB デバイスの実行に失敗しました。 再起動を行なうか、DPC-0401 の「IfUsbDevicePowerCtl」関数を使用し、USB デバイスの電源をOFF→ON して下さい。 「IfUsbDevicePowerCtl」関数の使用方法は、DPC-0401 のHelp を参照してください。
GPIB_ERROR_USB_TIMEOUT	-4001	USB デバイスとの通信がタイムアウトしました。 再起動を行なうか、DPC-0401 の「IfUsbDevicePowerCtl」関数を使用し、USB デバイスの電源をOFF→ON して下さい。 「IfUsbDevicePowerCtl」関数の使用方法は、DPC-0401 のHelp を参照してください。

4.5 プログラム例

4.5.1 高機能版DLL

C言語

```
//
================================================================================
//  GP-IB インタフェースモジュール用 Windows ドライバ
//  プログラム例 C 言語コンソールアプリケーション
//              (MS-DOS プロンプトまたはコマンド プロンプトで実行します)
//
//                         Copyright 2000, 2007 Interface Corporation. All rights
reserved.
//
//  プログラム説明
//  YOKOGAWA 7561 からの計測データを画面に表示します
//
================================================================================
#include <windows.h>
#include <stdio.h>
#include "GPC4304.H"

int  nInpDevAdrsTbl[2] = { 2, -1 };  // 入力機器アドレステーブル
char *szSendData = "F1R0SI100IT1";   // YOKOGAWA 7561 送信データ
char RecvBuf[100];                   // バッファ格納領域

int main(void)
{
    int   nRet;         // 関数戻り値
    DWORD nLen;         // バッファサイズ

    // ボードアクセス番号 0 のインタフェースモジュールを初期化
    nRet = PciGpibExInitBoard(0, 0);
    if (nRet) {
      printf("ボードアクセス番号 0 のインタフェースモジュールが初期化できません
¥n");
      printf("エラーコード : %d¥n", nRet);
      return -1;          // プログラム終了
    }

    // IFC を送出
    nRet = PciGpibExSetIfc(0, 1);
```

```
    if(nRet) {
      printf("IFC 送出に失敗しました¥n");
      printf("エラーコード : %d¥n", nRet);
      return -1;        //プログラム終了
    }

    // REN 信号有効
    nRet = PciGpibExSetRen(0);
    if(nRet) {
      printf("REN 信号有効に失敗しました¥n");
      printf("エラーコード : %d¥n", nRet);
      return -1;        //プログラム終了
    }

    // YOKOGAWA 7561 に対して次のモード設定を行います
    //
    //  ・DCV / AUTO RANGE / Interval100msec / 積分時間 2.5msec
    //
    printf("YOKOGAWA 7561 : 送信文字列 = %s¥n", szSendData);
    nRet    =    PciGpibExMastSendData(0,    nInpDevAdrsTbl,    strlen(szSendData),
szSendData, 0);
    if(nRet) {
      printf("データ送信に失敗しました¥n");
      printf("エラーコード : %d¥n", nRet);
      return -1;         //プログラム終了
    }

    // YOKOGAWA 7561 から測定データを受信します
    nLen = sizeof(RecvBuf);
    nRet = PciGpibExMastRecvData(0, nInpDevAdrsTbl, &nLen, RecvBuf, 0);
    if(nRet) {
      printf("データ受信に失敗しました¥n");
      printf("エラーコード : %d¥n", nRet);
      return -1;         //プログラム終了
    }

    // 受信文字列の表示
    printf("受信処理 戻り値 = %d¥n", nRet);
    printf("YOKOGAWA 7561 : 受信文字列 = %s¥n", RecvBuf);

    printf("YOKOGAWA 7561 : 受信データ長 = %d¥n", nLen);

    // ボードアクセス番号 0 のインタフェースモジュールを使用終了
    PciGpibExFinishBoard(0);
```

```
    return 0;
}
```

Visual Basic

```
'
================================================================================
=
'   GP-IB インタフェースモジュール用 Windows ドライバ
'   プログラム例  Visual Basic
'
'                        Copyright 2000, 2007 Interface Corporation. All rights
reserved.
'
'   プログラム説明
'   YOKOGAWA 7561 からの計測データを画面に表示します
'
================================================================================
=
' 変数定義
Dim nLen As Long              ' バッファサイズ
Dim szData As String          ' 送信データ
Dim RecvBuf As String         ' バッファ格納領域
Dim InpDevAdrsTbl(1) As Long  ' 入力機器アドレステーブル

' ボードアクセス番号 0 のインタフェースモジュールを初期化
nRet = PciGpibExInitBoard(0, 0)
If nRet <> 0 Then
    nRet=MsgBox("ボードアクセス番号 0 のインタフェースモジュールが初期化できません", _
                    (vbOKOnly + vbCritical), "PciGpibExInitBoard")
    Unload Me ' ダイアログを閉じる
End If

' IFC 送出
nRet = PciGpibExSetIfc(0, 1)
If nRet <> 0 Then
    nRet = MsgBox("IFC 送出に失敗しました", (vbOKOnly + vbCritical),
"PciGpibExSetIfc")
    PciGpibExFinishBoard (0) ' ボードアクセス番号 0 のインタフェースモジュールを使用終了
    Unload Me ' ダイアログを閉じる
End If

' REN 信号有効
nRet = PciGpibExSetRen(0)
If nRet <> 0 Then
    nRet = MsgBox("REN 信号有効に失敗しました", _
```

```
                              (vbOKOnly + vbCritical), "PciGpibExSetRen")
    PciGpibExFinishBoard (0) ' ボードアクセス番号0のインタフェースモジュールを使
用終了
    Unload Me ' ダイアログを閉じる
End If

' 機器アドレスを設定
InpDevAdrsTbl(0) = 2
InpDevAdrsTbl(1) = -1


' YOKOGAWA 7561 に対して次のモード設定を行います
'
'  ・DCV / AUTO RANGE / Interval100msec / 積分時間 2.5msec
'
' 送信文字列の表示
nRet = MsgBox("YOKOGAWA 7561 : 送信文字列 = F1R0SI100IT1", _
                  (vbOKOnly + vbInformation), "PciGpibExMastSendData")
szData = StrConv("F1R0SI100IT1", vbFromUnicode)
nLen = LenB(szData)
szData = StrConv(szData, vbUnicode)
nRet = PciGpibExMastSendData(0, InpDevAdrsTbl(0), nLen, szData, 0)
If nRet <> 0 Then
    nRet = MsgBox("データ送信に失敗しました", (vbOKOnly + vbCritical), _
                                                  "PciGpibExMastSendData")
    PciGpibExFinishBoard (0) ' ボードアクセス番号0のインタフェースモジュールを使
用終了
    Unload Me ' ダイアログを閉じる
End If

' YOKOGAWA 7561 から測定データを受信します
nLen = 32
RecvBuf = String(nLen, 0)
nRet = PciGpibExMastRecvData(0, InpDevAdrsTbl(0), nLen, RecvBuf, 0)
If nRet <> 0 Then
    nRet = MsgBox("データ受信に失敗しました", (vbOKOnly + vbCritical), _

"PciGpibExMastRecvData")
    PciGpibExFinishBoard (0) ' ボードアクセス番号0のインタフェースモジュールを使
用終了
    Unload Me ' ダイアログを閉じる
End If<

' 受信文字列の表示
nRet = MsgBox("YOKOGAWA 7561 : 受信文字列 = " & RecvBuf, (vbOKOnly + vbInformation),
```

```
_
"PciGpibExMastRecvData")

' ボードアクセス番号0のインタフェースモジュールを使用終了
PciGpibExFinishBoard (0)
```

Delphi

```
//
===============================================================================
//  GP-IB インタフェースモジュール用 Windows ドライバ
//  プログラム例 Delphi
//
//                        Copyright 2000, 2007 Interface Corporation. All rights
reserved.
//
//  プログラム説明
//  YOKOGAWA 7561 からの計測データを画面に表示します
//
===============================================================================
var
    nRet : Integer;                             // 関数戻り値
    gdwSLength : Cardinal;                      // 送信データ長
    gdwRLength : Cardinal;                      // 受信データ長
    pSendData : pChar;                          // 送信データ
    gnAdrsTbl : Array[0..7] of Integer;         // アドレステーブル
    gRecvBuf : Array[0..128] of Char;           // 受信バッファ

begin
    // ボードアクセス番号 0 のインタフェースモジュールを初期化
    nRet := PciGpibExInitBoard(0, 0);
    if nRet<>0 then
    begin
      MessageDlg('ボードアクセス番号 0 のインタフェースモジュールが初期化できませ
ん',
                                                   mtInformation, [mbOK], 0);
      Exit;      // ダイアログを閉じる
    end;

    // IFC を送出
    nRet := PciGpibExSetIfc(0, 1);
    if nRet<>0 then
    begin
      MessageDlg('IFC 送出に失敗しました', mtInformation, [mbOK], 0);
      PciGpibExFinishBoard(0);    // ボードアクセス番号 0 のインタフェースモジュー
ルを使用終了
      Exit;      // ダイアログを閉じる
    end;

    // REN 信号有効
    nRet := PciGpibExSetRen(0);
```

```
    if nRet<>0 then
    begin
      MessageDlg('REN 信号有効に失敗しました', mtInformation, [mbOK], 0);
      PciGpibExFinishBoard(0); // ボードアクセス番号 0 のインタフェースモジュール
を使用終了
      Exit;      // ダイアログを閉じる
    end;

    // 送信文字列設定
    pSendData := StrNew('F1R0SI100IT1');

    // 機器アドレス設定
    gnAdrsTbl[0] := 2;
    gnAdrsTbl[1] := -1;

    // YOKOGAWA 7561 に対して次のモード設定を行います
    //
    //  ・DCV / AUTO RANGE / Interval100msec / 積分時間 2.5msec
    //
    gdwSLength := 12;
    nRet  :=  PciGpibExMastSendData(0,  @gnAdrsTbl[0],  gdwSLength,  pSendData,
WM_NULL);

    StrDispose(pSendData);
    MessageDlg('YOKOGAWA 7561 : 送信文字列 = ' + 'F1R0SI100IT1',
                                                    mtInformation, [mbOK], 0);
    if nRet<>0 then
    begin
      MessageDlg('データ送信に失敗しました', mtInformation, [mbOK], 0);
      PciGpibExFinishBoard(0);   // ボードアクセス番号 0 のインタフェースモジュー
ルを使用終了
      Exit;      // ダイアログを閉じる
    end;

    // YOKOGAWA 7561 から測定データを受信します
    gdwRLength := 32;
    nRet  := PciGpibExMastRecvData(0,  @gnAdrsTbl[0],  gdwRLength,  @gRecvBuf[0],
WM_NULL);

    if nRet<>0 then
    begin
      MessageDlg('データ受信に失敗しました', mtInformation, [mbOK], 0);
      PciGpibExFinishBoard(0);   // ボードアクセス番号 0 のインタフェースモジュー
ルを使用終了
      Exit;      // ダイアログを閉じる
```

```
    end;

    // 受信文字列の表示
    MessageDlg('YOKOGAWA 7561 : 受信文字列 = ' + gRecvBuf, mtInformation, [mbOK],
0);
    MessageDlg('YOKOGAWA 7561 : 受信データ長 = ' + IntToStr(gdwRLength),
                                                     mtInformation,  [mbOK],
0);

    PciGpibExFinishBoard(0);    //ボードアクセス番号0のインタフェースモジュール
を使用終了
End.
```

4.5.2 標準版DLL

C言語

```
//
================================================================================
//  GP-IBインタフェースモジュール用Windowsドライバ
//  標準版DLLプログラム例   C言語コンソールアプリケーション
//                          (MS-DOS プロンプトまたはコマンド プロンプトで実行し
ます)
//
//                      Copyright 2000, 2007 Interface Corporation. All rights
reserved.
//
//  プログラム説明
//  YOKOGAWA 7561 からの計測データを画面に表示します
//
================================================================================
#include <windows.h>
#include <stdio.h>
#include "GPC43042.H"

int  nInpDevAdrsTbl[2] = { 2, -1 };  // 入力機器アドレステーブル
char *szSendData = "F1R0SI100IT1";   // YOKOGAWA 7561 送信データ
char RecvBuf[100];                   // バッファ格納領域

void main(void)
{
    int nRet;                    // 関数戻り値
    DWORD nLen;                  // バッファサイズ
    ULONG ulBoardNo = 0;         // ボード番号

    // ボード番号0 のインタフェースモジュールを初期化
    nRet = GpibOpen( ulBoardNo );
    if (nRet) {
        printf("ボード番号0のインタフェースモジュールが初期化できません¥n");
        printf("エラーコード : %d¥n", nRet);
        exit(0);                          // プログラム終了
    }

    // IFCを送出
    nRet = GpibSetIfc( ulBoardNo );
    if (nRet) {
        printf("IFC送出に失敗しました¥n");
```

```
        printf("エラーコード : %d¥n", nRet);
        GpibClose( ulBoardNo );             // ボード番号０のインタフェースモジュ
ールを使用終了
        exit(0);                            // プログラム終了
    }

    // REN 信号有効
    nRet = GpibSetRen( ulBoardNo );
    if (nRet) {
        printf("REN 信号有効に失敗しました¥n");
        printf("エラーコード : %d¥n", nRet);
        GpibClose( ulBoardNo );             // ボード番号０のインタフェースモジュ
ールを使用終了
        exit(0);                            // プログラム終了
    }

    // YOKOGAWA 7561 に対して次のモード設定を行います
    //
    // ・DCV / AUTO RANGE / Interval100msec / 積分時間 2.5msec
    //
    printf("YOKOGAWA 7561 : 送信文字列 = %s¥n", szSendData);
    nRet = GpibSend( ulBoardNo, nInpDevAdrsTbl, strlen(szSendData), szSendData );
    if(nRet) {
        printf("データ送信に失敗しました¥n");
        printf("エラーコード : %d¥n", nRet);
        GpibClose( ulBoardNo );             // ボード番号０のインタフェースモジュ
ールを使用終了
        exit(0);                            // プログラム終了
    }

    // YOKOGAWA 7561 から測定データを受信します
    nLen = sizeof(RecvBuf);
    nRet = GpibReceive( ulBoardNo, nInpDevAdrsTbl, &nLen, RecvBuf );
    if(nRet) {
        printf("データ受信に失敗しました¥n");
        printf("エラーコード : %d¥n", nRet);
        GpibClose( ulBoardNo );             // ボード番号０のインタフェースモジュ
ールを使用終了
        exit(0);                            // プログラム終了
    }

    // 受信文字列の表示
    printf("受信処理 戻り値 = %d¥n", nRet);
    printf("YOKOGAWA 7561 : 受信文字列 = %s¥n", RecvBuf);
    printf("YOKOGAWA 7561 : 受信データ長 = %d¥n", nLen);
```

```
    // ボード番号 0 のインタフェースモジュールを使用終了
    GpibClose( ulBoardNo );
}
```

Visual Basic

```
'
================================================================================
=
'   GP-IB インタフェースモジュール用 Windows ドライバ
'   標準版 DLL プログラム例  Visual Basic
'
'                        Copyright 2000, 2007 Interface Corporation. All rights
reserved.
'
'   プログラム説明
'   YOKOGAWA 7561 からの計測データを画面に表示します
'
================================================================================
=

' 変数定義
Dim nLen As Long                  ' バッファサイズ
Dim szData As String              ' 送信データ

Dim RecvBuf As String             ' バッファ格納領域
Dim InpDevAdrsTbl(1) As Long      ' 入力機器アドレステーブル
Dim ulBoardNo As Long

ulBoardNo = 0

' ボード番号 0 のインタフェースモジュールを初期化
nRet = GpibOpen( ulBoardNo )
If nRet <> 0 Then
    nRet = MsgBox("ボード番号 0 のインタフェースモジュールが初期化できません", _
                                            (vbOKOnly    +        vbCritical),
"GpibOpen")
    Unload Me ' ダイアログを閉じる
End If

' IFC 送出
nRet = GpibSetIfc( ulBoardNo )
If nRet <> 0 Then
    nRet = MsgBox("IFC 送出に失敗しました", (vbOKOnly + vbCritical), "GpibSetIfc")
    GpibClose( ulBoardNo ) ' ボード番号 0 のインタフェースモジュールを使用終了
    Unload Me ' ダイアログを閉じる
End If

' REN 信号有功
```

```
nRet = GpibSetRen( ulBoardNo )
If nRet <> 0 Then
    nRet = MsgBox("REN 信号有効に失敗しました", (vbOKOnly + vbCritical),
"GpibSetRen")
    GpibClose( ulBoardNo ) ' ボード番号0のインタフェースモジュールを使用終了
    Unload Me ' ダイアログを閉じる
End If

' 機器アドレスを設定
InpDevAdrsTbl(0) = 2
InpDevAdrsTbl(1) = -1

' YOKOGAWA 7561 に対して次のモード設定を行います
'
'  ・DCV / AUTO RANGE / Interval100msec / 積分時間 2.5msec
'
' 送信文字列の表示
nRet = MsgBox("YOKOGAWA 7561 : 送信文字列 = F1R0SI100IT1", _
                                            (vbOKOnly      +      vbInformation),
"GpibSend")
szData = StrConv("F1R0SI100IT1", vbFromUnicode)
nLen = LenB(szData)
szData = StrConv(szData, vbUnicode)
nRet = GpibSend( ulBoardNo, InpDevAdrsTbl(0), nLen, szData )
If nRet <> 0 Then
    nRet = MsgBox("データ送信に失敗しました", (vbOKOnly + vbCritical), "GpibSend")
    GpibClose( ulBoardNo ) ' ボード番号0のインタフェースモジュールを使用終了
    Unload Me ' ダイアログを閉じる
End If

' YOKOGAWA 7561 から測定データを受信します
nLen = 32
RecvBuf = String(nLen, 0)
nRet = GpibReceive( ulBoardNo, InpDevAdrsTbl(0), nLen, RecvBuf )
If nRet <> 0 Then
    nRet = MsgBox("データ受信に失敗しました", (vbOKOnly + vbCritical),
"GpibReceive")
    GpibClose( ulBoardNo ) ' ボード番号0のインタフェースモジュールを使用終了
    Unload Me ' ダイアログを閉じる
End If

' 受信文字列の表示
nRet = MsgBox("YOKOGAWA 7561 : 受信文字列 = " & RecvBuf, (vbOKOnly + vbInformation),
"GpibReceive")
```

```
’ボード番号０のインタフェースモジュールを使用終了
GpibClose( ulBoardNo )
```

Delphi

```
//
================================================================================
=
// GP-IB インタフェースモジュール用 Windows ドライバ
// 標準版 DLL プログラム例 Delphi
//
//                        Copyright 2000, 2007 Interface Corporation. All rights
reserved.
//
// プログラム説明
// YOKOGAWA 7561 からの計測データを画面に表示します
//
================================================================================
=
var
    nRet : Integer;                         // 関数戻り値
    gdwSLength : Cardinal;                  // 送信データ長
    gdwRLength : Cardinal;                  // 受信データ長
    pSendData : pChar;                      // 送信データ
    gnAdrsTbl : Array[0..7]of Integer;      // アドレステーブル
    gRecvBuf : Array[0..128]of Char;        // 受信バッファ
    ulBoardNo : Cardinal;
begin
    ulBoardNo := 0;

    // ボード番号 0 のインタフェースモジュールを初期化
    nRet := GpibOpen( ulBoardNo );
    if nRet <> 0 then
    begin
        MessageDlg('ボード番号 0 のインタフェースモジュールが初期化できません',
                   mtInformation, [mbOK], 0);
        Exit;          // ダイアログを閉じる
    end;

    // IFC を送出
    nRet := GpibSetIfc( ulBoardNo );
    if nRet <> 0 then
    begin
        MessageDlg('IFC 送出に失敗しました', mtInformation, [mbOK], 0);
        GpibClose( ulBoardNo );   // ボード番号 0 のインタフェースモジュールを使用
終了
        Exit;                               // ダイアログを閉じる
    end;
```

```
    // REN 信号有効
    nRet := GpibSetRen( ulBoardNo );
    if nRet <> 0 then
    begin
        MessageDlg('REN 信号有効に失敗しました', mtInformation, [mbOK], 0);
        GpibClose( ulBoardNo );   // ボード番号 0 のインタフェースモジュールを使用
終了
        Exit;                             // ダイアログを閉じる
    end;

    // 送信文字列設定
    pSendData := StrNew('F1R0SI100IT1');

    // 機器アドレス設定
    gnAdrsTbl[0] := 2;
    gnAdrsTbl[1] := -1;

    // YOKOGAWA 7561 に対して次のモード設定を行います
    //
    // ・DCV / AUTO RANGE / Interval100msec / 積分時間 2.5msec
    //
    gdwSLength := 12;
    nRet := GpibSend( ulBoardNo, @gnAdrsTbl[0], gdwSLength, pSendData );

    StrDispose(pSendData);
    MessageDlg('YOKOGAWA 7561 : 送信文字列 = ' + 'F1R0SI100IT1', mtInformation,
                                                                        [mbOK],
0);
    if nRet<>0 then
    begin
        MessageDlg('データ送信に失敗しました', mtInformation, [mbOK], 0);
        GpibClose( ulBoardNo );            // ボード番号 0 のインタフェースモジュ
ールを使用終了
        Exit;                              // ダイアログを閉じる
    end;

    // YOKOGAWA 7561 から測定データを受信します
    gdwRLength := 32;
    nRet := GpibReceive( ulBoardNo, @gnAdrsTbl[0], gdwRLength, @gRecvBuf[0] );

    if nRet<>0 then
    begin
        MessageDlg('データ受信に失敗しました', mtInformation, [mbOK], 0);
        GpibClose( ulBoardNo );   // ボード番号 0 のインタフェースモジュールを使用
```

```
終了
        Exit;                              // ダイアログを閉じる
    end;

    // 受信文字列の表示
    MessageDlg('YOKOGAWA 7561 : 受信文字列 = ' + gRecvBuf, mtInformation, [mbOK],
0);
    MessageDlg('YOKOGAWA 7561 : 受信データ長 = ' + IntToStr(gdwRLength),
                                                     mtInformation,  [mbOK],
0);

    GpibClose( ulBoardNo );       // ボード番号０のインタフェースモジュールを使用
終了
End.
```

4.6 マルチラインインタフェースメッセージ一覧

h:16進数, d:10進数, A:ASCII, M:インタフェースメッセージ

h	d	A	M
00	0	NUL	-
01	1	SOH	GTL
02	2	STX	-
03	3	ETX	-
04	4	EOT	SDC
05	5	ENQ	PPC
06	6	ACK	-
07	7	BEL	-
08	8	BS	GET
09	9	HT	TCT
0A	10	LF	-
0B	11	VT	-
0C	12	FF	-
0D	13	CR	-
0E	14	SO	-
0F	15	SI	-
10	16	DLE	-
11	17	DC1	LLO
12	18	DC2	-
13	19	DC3	-
14	20	DC4	DCL
15	21	NAK	PPU
16	22	SYN	-
17	23	ETB	-
18	24	CAN	SPE
19	25	EM	SPD
1A	26	SUB	-
1B	27	ESC	-
1C	28	FS	-
1D	29	GS	-
1E	30	RS	-
1F	31	US	-
20	32	SP	MLA00
21	33	!	MLA01
22	34	“	MLA02
23	35	#	MLA03
24	36	$	MLA04
25	37	%	MLA05
26	38	&	MLA06
27	39	‘	MLA07
28	40	(	MLA08
29	41	)	MLA09
2A	42	*	MLA10

h	d	A	M
2B	43	+	MLA11
2C	44	,	MLA12
2D	45	-	MLA13
2E	46	.	MLA14
2F	47	/	MLA15
30	48	0	MLA16
31	49	1	MLA17
32	50	2	MLA18
33	51	3	MLA19
34	52	4	MLA20
35	53	5	MLA21
36	54	6	MLA22
37	55	7	MLA23
38	56	8	MLA24
39	57	9	MLA25
3A	58	:	MLA26
3B	59	;	MLA27
3C	60	<	MLA28
3D	61	=	MLA29
3E	62	>	MLA30
3F	63	?	UNL
40	64	@	MTA00
41	65	A	MTA01
42	66	B	MTA02
43	67	C	MTA03
44	68	D	MTA04
45	69	E	MTA05
46	70	F	MTA06
47	71	G	MTA07
48	72	H	MTA08
49	73	I	MTA09
4A	74	J	MTA10
4B	75	K	MTA11
4C	76	L	MTA12
4D	77	M	MTA13
4E	78	N	MTA14
4F	79	O	MTA15
50	80	P	MTA16
51	81	Q	MTA17
52	82	R	MTA18
53	83	S	MTA19
54	84	T	MTA20
55	85	U	MTA21

h	d	A	M
56	86	V	MTA22
57	87	W	MTA23
58	88	X	MTA24
59	89	Y	MTA25
5A	90	Z	MTA26
5B	91	[	MTA27
5C	92	¥	MTA28
5D	93	]	MTA29
5E	94	^	MTA30
5F	95	_	UNT
60	96	`	MSA00,PPE
61	97	a	MSA01,PPE
62	98	b	MSA02,PPE
63	99	c	MSA03,PPE
64	100	d	MSA04,PPE
65	101	e	MSA05,PPE
66	102	f	MSA06,PPE
67	103	g	MSA07,PPE
68	104	h	MSA08,PPE
69	105	i	MSA09,PPE
6A	106	j	MSA10,PPE
6B	107	k	MSA11,PPE
6C	108	l	MSA12,PPE
6D	109	m	MSA13,PPE
6E	110	n	MSA14,PPE
6F	111	o	MSA15,PPE
70	112	p	MSA16,PPD
71	113	q	MSA17,PPD
72	114	r	MSA18,PPD
73	115	s	MSA19,PPD
74	116	t	MSA20,PPD
75	117	u	MSA21,PPD
76	118	v	MSA22,PPD
77	119	w	MSA23,PPD
78	120	x	MSA24,PPD
79	121	y	MSA25,PPD
7A	122	z	MSA26,PPD
7B	123	{	MSA27,PPD
7C	124	|	MSA28,PPD
7D	125	}	MSA29,PPD
7E	126	~	MSA30,PPD
7F	127	DEL	-

■マルチラインインタフェースメッセージの定義

PPD　：　Parallel Poll Disable（パラレルポールの設定解除）
PPE　：　Parallel Poll Enable（パラレルポールの設定）
PPU　：　Parallel Poll Unconfigure（パラレルポール用設定の解除）
SDC　：　Selected Device Clear（選択されたデバイスのクリア）
SPD　：　Serial Poll Disable（シリアルポールの設定解除）
SPE　：　Serial Poll Enable（シリアルポールの設定）
TCT　：　Take Control（コントロールの取得）
UNL　：　Unlisten（全リスナの解除）
UNT　：　Untalk（全トーカの解除）
DCL　：　Device Clear（デバイスのクリア）
GET　：　Group Execute Trigger（グループトリガ実行）
GTL　：　Go To Local（ローカルモードに設定）
LLO　：　Local Lockout（ローカルのロックアウト）
MLA　：　My Listen Address（リスンアドレス）
MSA　：　My Secondary Address（2次アドレス）
MTA　：　My Talk Address（トーカアドレス）
PPC　：　Parallel Poll Configure（パラレルポールの設定）

第5章　サンプルプログラム

サンプルプログラムには実行形式のファイルが付属していません。
ソースコードをコンパイルして実行ファイルを生成してから、動作させてください。

●Visual C++の場合
Visual C++（Visual Studio）を起動し、「ファイル」メニューから「ワークスペースを開く」を選び、メイクファイル「*.mak」を開き、ビルドしてください。

●Visual Basicの場合
Visual Basicを起動し、プロジェクトファイル「*.vbp」を開き、ビルドしてください。

●Delphiの場合
Delphiを起動し、プロジェクトファイル「*.dpr」を開き、ビルドしてください。
作成後、「*.exe」を起動してください。

5.1　高機能版DLL

以下、高機能版DLLの各サンプルプログラムの概要を説明します。

プログラム名	内容
Async_X（※）	非同期送受信を行います。
Binary_X（※）	バイナリデータの送受信を行います。
Ctrl_X（※）	計測機器の制御を行います。
Device_X（※）	計測機器(YOKOGAWA7561、HP3245A)の制御を行います。
ManualOp_X（※）	関数の手動実行を行います。
MultiDevice_X（※）	複数機器の制御を行います。
csend	コントローラの送信を行います。
creceive	コントローラの受信を行います。
spoll	シリアルポールを行います。
tsend	トーカ（非コントローラ）の送信を行います。
lreceive	リスナ（非コントローラ）の受信を行います。
srq	サービスリクエストを行います。

※プログラム名のXは各言語により異なります (X = C:Visual C++, B:Visual Basic, D:Delphi)

5.2　標準版DLL

以下、標準版DLLの各サンプルプログラムの概要を説明します。

プログラム名	内容
Ctrl_X2（※）	計測機器の制御を行います。

※プログラム名のXは各言語により異なります (X = C:Visual C++, B:Visual Basic, D:Delphi)

第6章 ユーティリティ

以下、各ユーティリティの概要を説明します。

6.1 動作確認プログラム

6.1.1 GP-IBユーティリティ

GP-IBの動作を確認するためのGUIプログラムを用意しています。

【起動方法】

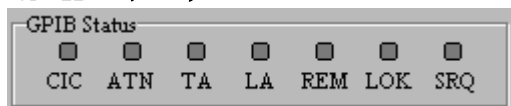
付属ソフトウェアのインストール完了後、「スタート」メニューより「Interface GPC-4301」→「GP-IB ユーティリティプログラム」を起動します。

【メニュー】

メニュー	プルダウンメニュー	内容
ファイル	ボードのオープン	GP-IB インタフェースモジュールを使用するためにオープンを行います。 以下のような設定用ダイアログボックスが表示されますので、 オープンするインタフェースモジュールとアドレスとデリミタを選択し「OK」を押します。 ボードのオープン ボード PCI-4301(RSW1設定値:0) プライマリアドレス 00 h セカンダリアドレス h デリミタ EOI CR+EOI LF CRLF LF+EOI CRLF+EOI CR OK Cancel ソルコン Classembly Devices®、I/O 付きタッチパネル Classembly Devices®の GP-IB モデルをお使いの場合、「PCI-432601」と表示されます。
	ボードのクローズ	オープンされている GP-IB インタフェースモジュールをクローズします。
	アプリケーションの終了	GP-IB ユーティリティを終了させます。

メニュー	プルダウンメニュー	内容
編集	バッファのクリア	送信データ入力テキストボックス、受信データ表示テキストボックス、バスコマンド入力ボックスをクリアします。
ヘルプ	バージョン情報	GP-IB ユーティリティのバージョン情報を表示します。

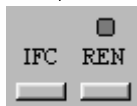
【画面の説明】

■ GP-IB ステータス

ステータス	説明
CIC(Controller in Charge)	コントローラ・イン・チャージであることを表します。
ATN(Attention)	ATN ラインの状態を表します。
TA(Talk Active)	トーカであることを表します。
LA(Listen Active)	リスナであることを表します。
REM(Remote)	リモート状態を表します。
LOK(Lockout)	ロックアウト状態を表します。
SRQ(Service Request)	SRQ ラインの状態を表します。

■ ボードアドレス

ボードアドレス Pri:01h 2nd:62h

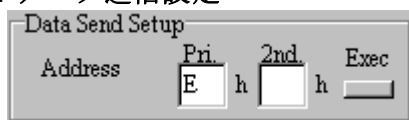
使用するインタフェースモジュールのアドレスを表します。
2 次アドレスを使用しない場合は、1 次アドレスのみ表示します。

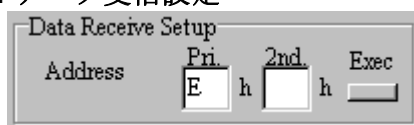
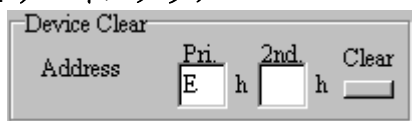
■ IFC/REN ボタン

IFC ボタン：IFC 信号を 100μs 送出します。
REN ボタン：REN 信号の ON/OFF を行います。
REN 信号が Assert 状態のとき上の LED が点灯します。

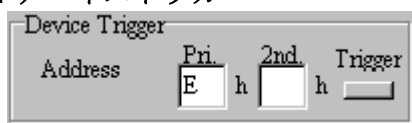
■ モード設定

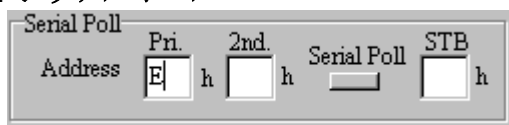
ユーティリティのモードを設定します
MASTER モードのときのみ有効です(MASTER モードへの切り替えはコントロールパネルで行います)。

モード	説明
Data Send	データ送信を行います
Data Receive	データ受信を行います
Device Clear	デバイスクリアを行います
Device Trigger	デバイストリガを行います
Serial Poll	シリアルポールを行います
Bus Command	バスコマンドを送信します

■ データ送信設定

送信セットアップを行います。
「Pri.」ボックスに接続機器の 1 次アドレス、「2nd.」ボックスに 2 次アドレスを入力し「Exec」ボタンを押します。
(2 次アドレスがない場合は「2nd.」ボックスには何も入力しません)
ボタンを押すと送信モードに切り替わり、送信データ入力テキストボックスが開かれ、設定した接続相手に対してデータの送信が行えるようになります。
「Exec」ボタンをもう一度押すと送信データ入力テキストボックスは閉じ送信モードを解除します。

■ データ受信設定

受信セットアップを行います。
「Pri.」ボックスに接続機器の 1 次アドレス、「2nd.」ボックスに 2 次アドレスを入力し「Exec」ボタンを押します。
(2 次アドレスがない場合は「2nd.」ボックスには何も入力しません)
ボタンを押すと受信モードに切り替わり、受信データ表示テキストボックスが開かれ、設定した接続相手からデータの受信が行えるようになります。
「Exec」ボタンをもう一度押すと受信データ表示テキストボックスは閉じ受信モードを解除します。

■ デバイスクリア

デバイスクリアコマンドを送信します。
「Pri.」ボックスに送信する接続機器の 1 次アドレス、「2nd.」ボックスに 2 次アドレスを入力し「Clear」ボタンを押します。
(2 次アドレスがない場合は「2nd.」ボックスには何も入力しません)

■ デバイストリガ

デバイストリガコマンドを送信します。
「Pri.」ボックスに送信する接続機器の 1 次アドレス、「2nd.」ボックスに 2 次アドレスを入力し「Trigger」ボタンを押します。
(2 次アドレスがない場合は「2nd.」ボックスには何も入力しません)

■ シリアルポール

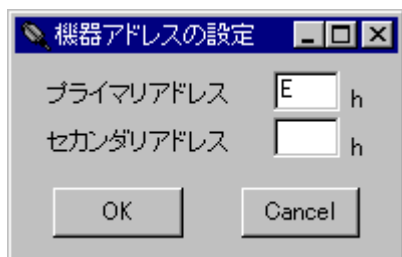
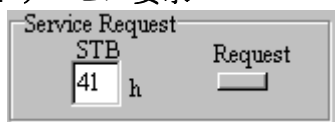
シリアルポールを行います。
「Pri.」ボックスにシリアルポールを行う接続機器の 1 次アドレス、「2nd.」ボックスに 2 次アドレスを入力し「Serial Poll」ボタンを押します。
(2 次アドレスがない場合は「2nd.」ボックスには何も入力しません)
機器から受信した STB(Status Byte)は「STB」ボックスに表示されます。

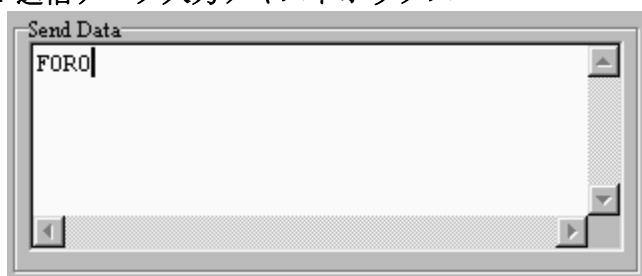
■ バスコマンド

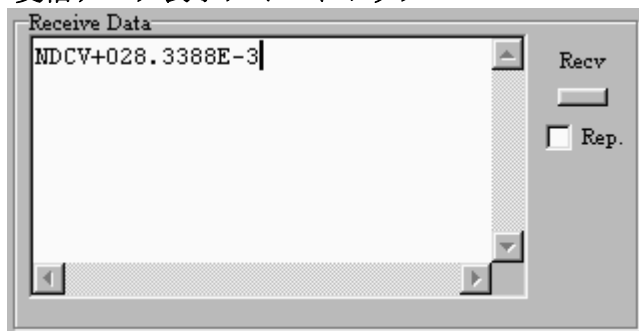
バスコマンドの送信を行います。
「Msg.」ボックス列にバスコマンドのコードを 16 進数で入力していきます(最大 10 バイト)
「Send」ボタンで入力したバスコマンドが送信されます

「Msg.」ボックス内でマウスを右クリックするとバスコマンドの一覧が表示されます(右図)
メニューを選択すると該当するコードが「Msg.」ボックスに入ります。
一覧表示されるバスコマンドは以下のとおりです。

バスコマンド	コード
GTL(Go to Local)	01h
SDC(Selected Device Clear)	04h
GET(Group Execute Trigger)	08h
LLO(Local Lockout)	11h
DCL(Device Clear)	14h
MTA(My Talk Address)	[弊社 GP-IB インタフェースの 1 次アドレス] + 40h [弊社 GP-IB インタフェースの 2 次アドレス](2 次アドレス設定時)
MLA(My Listen Address)	[弊社 GP-IB インタフェースの 1 次アドレス] + 20h [弊社 GP-IB インタフェースの 2 次アドレス](2 次アドレス設定時)
TA(Talk Address)	[接続機器の 1 次アドレス(※)] + 40h [接続機器の 2 次アドレス(※)](2 次アドレス設定時)
LA(Listen Address)	[接続機器の 1 次アドレス(※)] + 20h [接続機器の 2 次アドレス(※)](2 次アドレス設定時)

※接続機器のアドレスは以下のアドレス入力用ダイアログボックスで入力します。

■ サービス要求

SLAVE モードのときのみ有効です(SLAVE モードへの切り替えはコントロールパネルで行います)
「STB」ボックスに STB(Status Byte)を入力し「Request」ボタンを押します。
サービス要求が行われ SRQ ラインが有効になります。

■ 送信データ入力テキストボックス

送信するデータを入力します。
リターンキーを押した行のデータを接続機器に対して送信します。

■ 受信データ表示テキストボックス

受信したデータを表示します。
「Recv」ボタンを押すと機器から受け取ったデータを表示します
「Rep.」チェックボックスにチェックを入れておくと、連続してデータを受信し表示します。

【内部処理】

GP-IB ユーティリティ内部にて、どのようにインタフェースモジュールの制御を行ってい

るか、その概略を説明いたします。

1.メイン処理

1) PciGpibExInitBoard 関数にて初期化

2) PciGpibExSetConfig 関数にてパラメータ設定
→ここでインタフェースモジュールオープン時のデリミタ設定が行われます。

3) タイマをスタートさせる。
→後述の「4.タイマルーチン」がバックグラウンドにて常時動作している状態となります。

4) 入力待ち
→「Data Send」ボタン、「Data Receve」ボタンなどの入力待ちとなります。

5) 押されたボタンにより処理を選択

2.データ送信（Data Send）

1) 「Data Send Setup」にて以下の処理を行います。
1.PciGpibExGoActCtrller 関数により、ATN をアサート（コマンドモードへ遷移）
2.PciGpibExWriteBusCmd 関数にて UNL、MTA、LA 送信
→UNL…全ての機器のリスナ指定を解除（3Fh を送信）
MTA…インタフェースモジュール自身をトーカ指定（インタフェースモジュールのアドレスが 0 ならば、40h を送信）
LA …入力した機器アドレスにより機器をリスナ指定（機器アドレスが 1 ならば、21h
を送信）
3.PciGpibExGoStandby 関数により、ATN をデアサート（データモードへ遷移）

2) 送信入力ボックスが開きます。

3) 送信入力ボックスへ機器へ送信したいデータを入力
→ENTER キーを押すと、次のデータ送信へ

4) データ送信
→PciGpibExSlavSendData 関数を実行して、データ送信を行います。

3.データ受信（Data Receive）

1) 「Data Receive Setup」にて以下の処理を行います。
1.PciGpibExGoActCtrlle r 関数により、ATN をアサート（コマンドモードへ遷移）
2.PciGpibExWriteBusCmd 関数 にて UNL、MLA、TA 送信
→UNL…全ての機器のリスナ指定を解除（3Fh を送信）
MLA…インタフェースモジュール自身をリスナ指定（インタフェースモジュールのアドレスが 0 ならば、20h を送信）
TA …入力した機器アドレスより機器をトーカ指定（機器アドレスが 1 ならば、41h
を送信）
3. PciGpibExGoStandby 関数により、ATN をデアサート（データモードへ遷移）

2) 受信データ表示ボックスが開きます。
→「Recv」ボタンを押した場合には、3)へ
→「Rep.」チェックボックスをチェックした場合には、4)へ

3) データ受信
→PciGpibExSlavRecvData 関数を実行して、データ受信を行い、表示します。

4）連続データ送信
→データ受信スレッドを起動して､スレッド内にて繰り返し､PciGpibExSlavRecvData関数を実行
→「Rep.」ボタンのチェックが外されるまで、スレッドが動作し続けます。
→受信バッファは、16384バイトまでとなっており、バッファばフルになるまで受信し続
けます。

4. タイマルーチン
1）定期的にタイマイベントが発生してタイマルーチンが呼び出されています。
→周期 100ms
2）タイマルーチンでは、PciGpibExGetStatus 関数 を呼び出して、GP-IB Status の更新を行っています。

6.1.2 コマンドラインユーティリティ

【実行方法】

コマンドプロンプトを開き、「インストール指定先ディレクトリ¥bin」フォルダ内のGpibStartを実行してください。タスクトレイに、GP-IBアイコンが表示されます。

その後、下記の各コマンドラインプログラムを実行して下さい。
GP-IB制御を終了したら、「インストール指定先ディレクトリ¥bin」フォルダ内のGpibStopを実行してください。タスクトレイのGP-IBアイコンが削除されます。

【GP-IB動作確認プログラム】

GP-IBの動作を確認するための下記の5つのコマンドラインプログラムを用意しています。

プログラム名	内容
GpibClear	接続機器に対し、デバイスクリアを実行します。
GpibWrite	接続機器にデータを送信します。
GpibRead	接続機器からデータを受信します。
GpibTrigger	接続機器に対し、デバイストリガを実行します。
GpibSpoll	接続機器に対し、シリアルポールを実行します。

これらのプログラムで使用するパラメータは全て 16 進数指定ができます。その場合、0xの後に16進値を指定してください。
例）GpibClearでタイムアウトに100hを指定する場合

```
>GpibClear -t 0x100
```

【注意事項】

※ 各コマンドラインユーティリティは、「GpibStart」実行後でのみ動作します。
※ 各コマンドラインユーティリティは、コントローラでのみ使用できます。
あらかじめ『3.2　コントロールパネル』で、動作モードを「MASTER」に設定しておいてください。
※ コマンドラインでの制御終了時には、必ず「GpibStop」を実行してください。他のアプリケーションでGP-IBインタフェースの制御が行えなくなる可能性があります。

各ユーティリティは、正常終了しますと、0を返します。もし、正常に処理が終了しなかった場合、0以外の値を返します。0以外の値が返された場合については、『4.4　戻り値一覧』を参照してください。
また、エラー発生時は標準エラー出力に戻り値とエラー情報を出力します。

GpibClear正常終了時の例

```
>GpibClear
>echo %ERRORLEVEL%
0
```

GpibClearエラー時の例

```
>GpibClear -n 15
```

```
ERROR: -1: The specified board number is invalid.
```

■GpibClear

接続機器に対し、デバイスクリアを実行します。

【書式】

```
GpibClear [-t timeout] [-?] [-n boardno] [adrs]
```

【パラメータ】

-t *timeout*

省略可能オプションです。デバイスクリアが完了するまでのタイムアウト時間を100[ms]単位で設定します。例えば、10 を設定すると、100×10[ms]後にタイムアウトします。
省略した場合、『3.2　コントロールパネル』の「コマンド送出タイムアウト」の値が適用されます。
設定可能範囲は 1～65535 です。

-?

省略可能オプションです。GpibClear プログラムの書式を出力します。
本オプションを付けて実行した場合は、デバイスクリアは行わず、書式を出力して終了します。

-n *boardno*

インタフェースモジュール上のロータリスイッチ（RSW1）の設定値（CardBus 製品では CardBus ID 設定値）を指定します。省略可能で、省略した場合にはボード番号 0 を使用します。

adrs

デバイスクリアの対象となる機器のアドレスを指定します。
アドレスを指定した場合は、そのアドレスの機器に対して SDC（Selected Device Clear：選択したデバイスのクリア）を発行します。
本パラメータは省略可能で、省略した場合、DCL（全デバイスのクリア）を発行します。
2 次アドレスを持つ機器が対象の場合、1 次アドレスと 2 次アドレスをカンマ(,)で区切ってください。
例）1 次アドレスが 2 の機器が対象の場合：2
例）1 次アドレスが 2、2 次アドレスが 96 の機器が対象の場合：2,96

【使用例】

```
>GpibClear 2
```

ボード番号 0 の GP-IB インタフェースから、1 次アドレスが 2 の機器に対して、デバイスクリアを実行します。

■GpibWrite

接続機器にデータを送信します。
一度に送信できるデータは1024バイトまでです。送信データは””で囲んでください。””が付いていない場合、期待したデータが送信されない場合があります。

【書式】

`GpibWrite [-s size] [-t timeout] [-e delim] [-?] [-n boardno] adrs ["data"]`

【入力】

送信データを標準入力（stdin）から指定することができます（使用例参照）。

【パラメータ】

-s *size*

省略可能オプションです。送信データのバイトサイズを指定します。
指定した場合、指定したサイズ分だけデータを送信します。文字列終端までの全送信データのバイト数が指定したバイト数に収まる場合は、全送信データを送信します。
省略した場合、文字列終端までの全送信データを送信します（ただし、1024 バイトまで）。
設定可能範囲は 0～1024 です。0 を指定した場合、弊社 GP-IB インタフェースをトーカ、制御機器をリスナに指定して終了します。

-t *timeout*

省略可能オプションです。データ送信が完了するまでのタイムアウト時間を 100[ms]単位で設定します。例えば、10 を設定すると、100×10[ms]後にタイムアウトします。
省略した場合、『3.2　コントロールパネル』の「送受信タイムアウト」の値が適用されます。
設定可能範囲は 1～65535 です。

-e *delim*

省略可能オプションです。送信時に使用するデリミタを指定します。
下記の値から選択できます。

値	送信デリミタ
0	デリミタ無し
1	EOI
2	CR
3	CR+EOI
4	LF
5	LF+EOI
6	CRLF
7	CRLF+EOI
8	NULL

省略した場合、『3.2　コントロールパネル』の送信デリミタが適用されます。

-?

省略可能オプションです。GpibWrite プログラムの書式を出力します。
本オプションを付けて実行した場合は、データ受信は行わず、書式を出力して終了します。

-n *boardno*
インタフェースモジュール上のロータリスイッチ（RSW1）の設定値（CardBus 製品では CardBus ID 設定値）を指定します。省略可能で、省略した場合にはボード番号 0 を使用します。

adrs
データ受信先の機器のアドレスを指定します。
指定したアドレスの機器へデータを送信します。
2 次アドレスを持つ機器が対象の場合、1 次アドレスと 2 次アドレスをカンマ(,)で区切ってください。
例）1 次アドレスが 2 の機器が対象の場合：2
例）1 次アドレスが 2、2 次アドレスが 96 の機器が対象の場合：2,96

data
送信データを指定します。省略可能です。
省略した場合、標準入力からのデータ指定があれば、そのデータを送信します。
標準入力からのデータ指定がない場合は、プログラム実行後に送信データを入力して「Enter」キーを押すと、そのデータを送信します。

【使用例】

```
>GpibWrite -n 1 2 ″*RST″
```

ボード番号1のGP-IBインタフェースから、1次アドレスが2の機器へ”*RST”を送信します。

```
>echo ″*RST″ | GpibWrite 2
```

パイプを使って入力した”*RST”を、ボード番号 0 の GP-IB インタフェースから、1 次アドレスが 2 の機器へ送信します。

```
>GpibWrite 2
″*RST″
```

ボード番号0のGP-IBインタフェースから、1次アドレスが2の機器へ”*RST”を送信します。

```
>type data.txt | GpibWrite 2
```

パイプを使って入力した data.txt の中身を、ボード番号 0 の GP-IB インタフェースから、1 次アドレスが 2 の機器へ送信します。

■GpibRead

接続機器からデータを受信します。
一度に受信できるデータは1024バイトまでです。

【書式】

```
GpibRead [-t timeout] [-c count] [-e delim] [-?] [-n boardno] adrs
```

【出力】

データ受信が正常に終了した場合、受信したデータが標準出力（stdout）に出力されます（文字列出力）。

【パラメータ】

-t *timeout*

省略可能オプションです。データ受信が完了するまでのタイムアウト時間を100[ms]単位で設定します。例えば、10を設定すると、100×10[ms]後にタイムアウトします。
省略した場合、『3.2　コントロールパネル』の「送受信タイムアウト」の値が適用されます。
設定可能範囲は1～65535です。

-c *count*

省略可能オプションです。受信するデータのバイト数を指定します。
指定した場合、デリミタを受信するか、指定したバイト数までデータを受信した時点で、処理を終了します。
省略した場合、デリミタを受信するか、1024バイトのデータを受信した時点で、処理を終了します。
設定可能範囲は0～1024です。0を指定した場合、弊社GP-IBインタフェースをリスナ、制御機器をトーカに指定して終了します。

-e *delim*

省略可能オプションです。受信時に使用するデリミタを指定します。
下記の値から選択できます。

値	受信デリミタ
0	デリミタ無し
1	EOI
2	CR
3	CR+EOI
4	LF
5	LF+EOI
6	CRLF
7	CRLF+EOI
8	NULL

省略した場合、『3.2　コントロールパネル』の受信デリミタが適用されます。

-?

省略可能オプションです。GpibRead プログラムの書式を出力します。
本オプションを付けて実行した場合は、データ受信は行わず、書式を出力して終了します。

-n *boardno*
インタフェースモジュール上のロータリスイッチ（RSW1）の設定値（CardBus 製品では CardBus ID 設定値）を指定します。省略可能で、省略した場合にはボード番号 0 を使用します。

adrs
データ送信元の機器のアドレスを指定します。
指定したアドレスの機器からデータを受信します。
2 次アドレスを持つ機器が対象の場合、1 次アドレスと 2 次アドレスをカンマ(,)で区切ってください。
例）1 次アドレスが 2 の機器が対象の場合：2
例）1 次アドレスが 2、2 次アドレスが 96 の機器が対象の場合：2,96

【使用例】

```
>GpibRead 2
NDCV+028.3388E-3
```

ボード番号 0 の GP-IB インタフェースが、1 次アドレスが 2 の機器から”NDCV+028.3388E-3” のデータを受信しました。

```
>GpibRead 2 > log.txt
```

ボード番号 0 の GP-IB インタフェースが、1 次アドレスが 2 の機器からデータを読み込み、リダイレクトを使って log.txt ファイルに受信データを出力しました。

■GpibTrigger
接続機器に対し、デバイストリガを実行します。

【書式】

```
GpibTrigger [-t timeout] [-?] [-n boardno] adrs
```

【パラメータ】

-t timeout

省略可能オプションです。デバイストリガが完了するまでのタイムアウト時間を100[ms]単位で設定します。例えば、10 を設定すると、100×10[ms]後にタイムアウトします。
省略した場合、『3.2　コントロールパネル』の「コマンド送出タイムアウト」の値が適用されます。
設定可能範囲は 1～65535 です。

-?

省略可能オプションです。GpibTrigger プログラムの書式を出力します。
本オプションを付けて実行した場合は、デバイストリガは行わず、書式を出力して終了します。

-n *boardno*

インタフェースモジュール上のロータリスイッチ（RSW1）の設定値（CardBus 製品では CardBus ID 設定値）を指定します。省略可能で、省略した場合にはボード番号 0 を使用します。

adrs

デバイストリガの対象となる機器のアドレスを指定します。
指定したアドレスの機器に対して GET（Group Execute Trigger:グループトリガ実行）を発行します。
2 次アドレスを持つ機器が対象の場合、1 次アドレスと 2 次アドレスをカンマ(,)で区切ってください。
例）1 次アドレスが 2 の機器が対象の場合：2
例）1 次アドレスが 2、2 次アドレスが 96 の機器が対象の場合：2,96

【使用例】

```
>GpibTrigger 2
```

ボード番号 0 の GP-IB インタフェースから、1 次アドレスが 2 の機器に対して、デバイストリガを実行します。

■GpibSpoll

接続機器に対し、シリアルポールを実行します。

【書式】

```
GpibSpoll [-t timeout] [-?] [-n boardno] adrs
```

【出力】

シリアルポールが正常に終了した場合、取得したステータスバイト(STB)が 16 進数で標準出力（stdout）に出力されます。

【パラメータ】

-t timeout

省略可能オプションです。ステータスバイト受信が完了するまでのタイムアウト時間を 100[ms]単位で設定します。例えば、10 を設定すると、100×10[ms]後にタイムアウトします。
省略した場合、『3.2　コントロールパネル』の「STB応答時間」の値が適用されます。
設定可能範囲は 1～65535 です。

-?

省略可能オプションです。GpibSpoll プログラムの書式を出力します。
本オプションを付けて実行した場合は、シリアルポールは行わず、書式を出力して終了します。

-n *boardno*

インタフェースモジュール上のロータリスイッチ（RSW1）の設定値（CardBus 製品では CardBus ID 設定値）を指定します。省略可能で、省略した場合にはボード番号 0 を使用します。

adrs

デバイストリガの対象となる機器のアドレスを指定します。
指定したアドレスの機器に対して GET（Group Execute Trigger:グループトリガ実行）を発行します。
2 次アドレスを持つ機器が対象の場合、1 次アドレスと 2 次アドレスをカンマ(,)で区切ってください。
例）1 次アドレスが 2 の機器が対象の場合：2
例）1 次アドレスが 2、2 次アドレスが 96 の機器が対象の場合：2,96

【使用例】

```
>GpibSpoll 2
41
```

ボード番号 0 の GP-IB インタフェースから、1 次アドレスが 2 の機器に対して、シリアルポールを実行し、41h のステータスバイトを取得しました。

6.2 自己診断プログラム

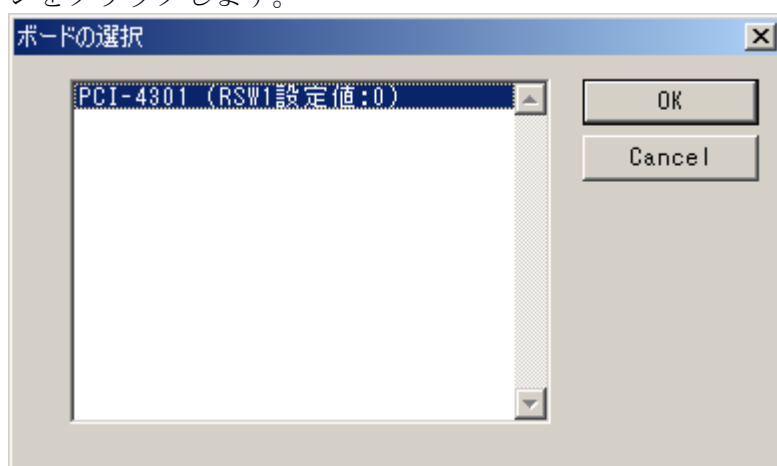
本製品には、動作不具合時の原因がハードウェア的なものか、ソフトウェア的なものかを容易に判断するための自己診断機能を搭載しています。診断プログラムを用いて動作確認を行ってください。
自己診断プログラムにより、以下のことが確認できます。
ソルコン Classembly Devices®、I/O付きタッチパネル Classembly Devices®のGP-IBモデルをお使いの場合は、本プログラムはお使いになれません。

●デバイスドライバがインストールされているか、また正常にインストール完了しているか
●コンピュータとGP-IBインタフェースモジュールとのインタフェース動作（レジスタアクセス，割り込み信号）

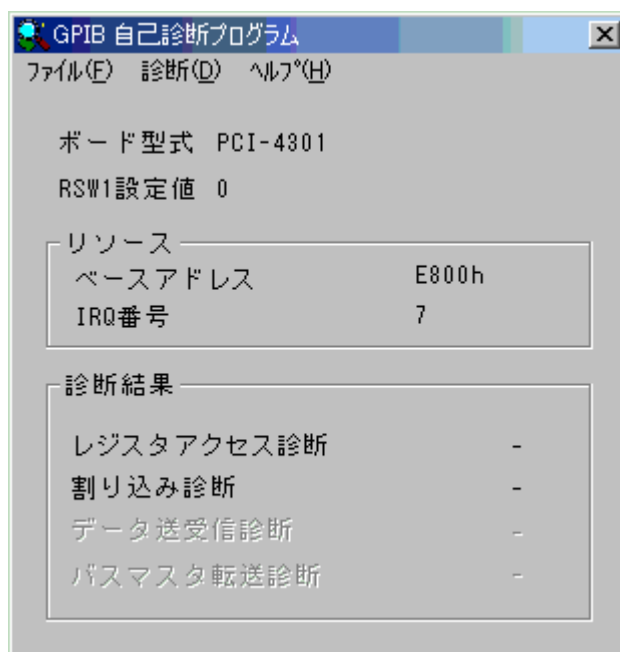
自己診断プログラムが正常に動作すれば、システムの不具合の原因を以下の要因に絞ることができます。

●GP-IBインタフェースモジュールのGP-IBインタフェース部分の故障
●GP-IBケーブルの破損
●接続しているGP-IB機器の動作不具合
●アプリケーションプログラムの不具合

【必要な機材】
・弊社 GP-IB インタフェース
・自己診断プログラム(DiagGpib.exe)

【起動方法】
1) PCI/CompactPCI ソフトウェアのインストール完了後、「スタート」メニューより「プログラム」-「Interface GPC-4301」-「DiagGpib」を起動します。
2) メニューの「ファイル」-「デバイスの選択」を選択すると、インストールされているデバイスの一覧が表示されます。その中から、使用したいデバイスを選択し、「OK」ボタンをクリックします。

インタフェースモジュールの一覧表示に、対象のインタフェースモジュールが見つからない場合は、以下のことが考えられます。

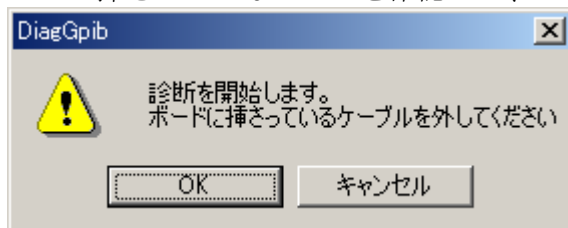
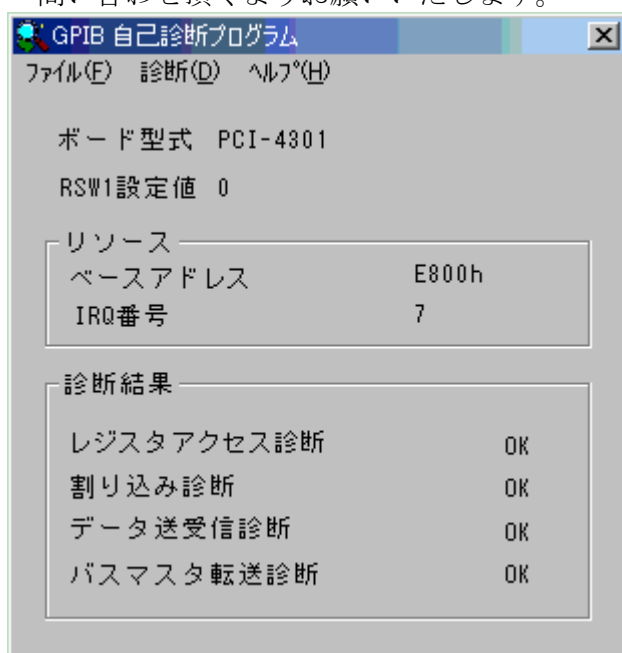
● デバイスドライバが正常にインストールされていない。
● コンピュータのBIOSや、Windowsによって、GP-IBインタフェースモジュールが正しく初期化されていない。
● GP-IBインタフェースモジュールの故障。弊社 カスタマーサポートセンタまでお問い合わせください。

検査を行うインタフェースモジュールを選択すると、画面に、型式名、RSW1の設定値、I/Oポート（またはメモリ）ベースアドレス、IRQ番号が表示されます。

コントロールパネル「Interface GP-IB Configuration」の設定で、インタフェースモジュールの動作モードは、必ず「MASTER」にしてください。「SLAVE」にすると正しく診断できません。 複数枚使用する場合には、RSW1の設定値をインタフェースモジュール毎で異なる値に設定してください。

※このユーティリティで同時に複数のデバイスをオープンすることはできません。

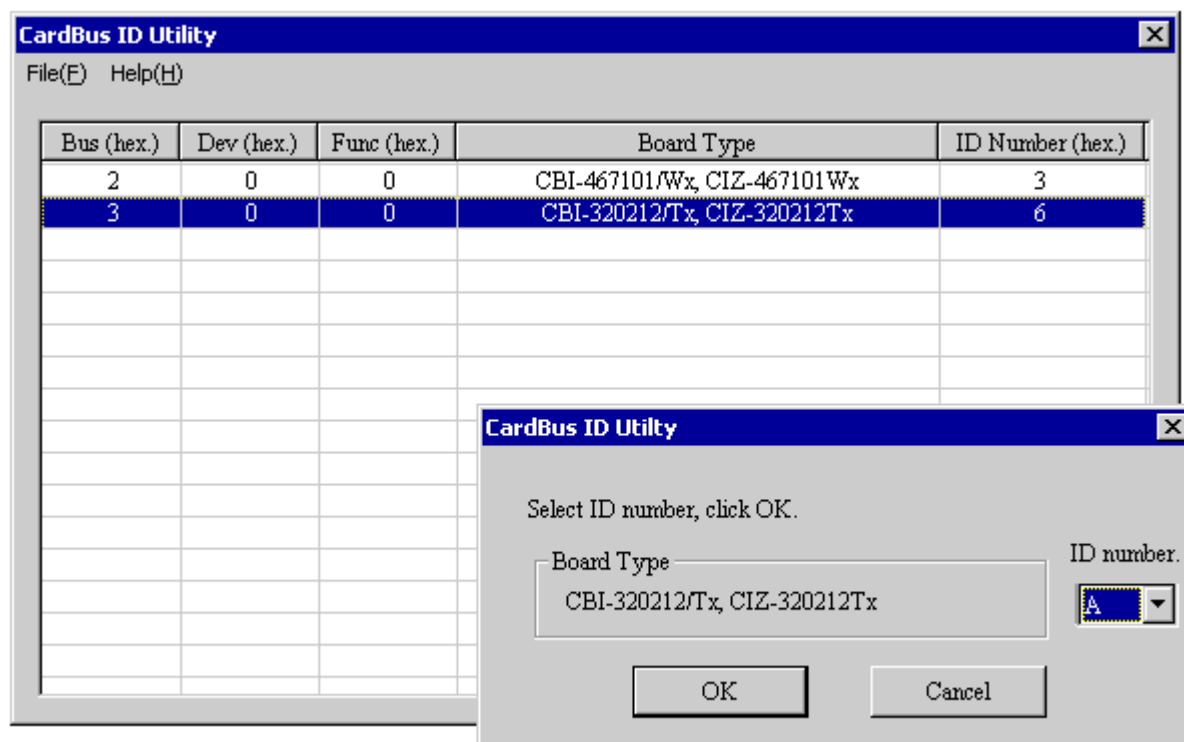
※GP-IB製品を複数枚使用する場合には、RSW1の設定値をインタフェースモジュールごとで異なる値に設定してください。

【診断方法】

1) メニューの「診断」－「診断開始」を選択します。
2) 診断開始のダイアログボックスが表示されますので、インタフェースモジュールにケーブルが挿さっていないことを確認して、OK ボタンをクリックします。

3) 自動的に各項目の診断が行われ、結果が表示されます。
4) 診断結果が「NG」の場合には、メニューから「ファイル」－「結果の保存」で診断結果を印刷してください。
   その診断結果とともに弊社カスタマーサポートセンタまでお問い合わせください。
   ※動作モードが「MASTER」であること、ケーブルが接続されていないことを確認のうえ、問い合わせ頂くようお願いいたします。

6.3 CardBus ID設定ユーティリティ

このユーティリティは複数枚のGP-IB製品を使用するためのものです。製品ごとに異なるID番号を設定します。このID番号はPCI インタフェースモジュールで言えばRSW番号と同等の意味を持ち、デバイスマネージャ、自己診断プログラムでも、RSW1として表示されます。

【操作方法】

1. ドライバソフトウェアのインストール完了後、「コントロールパネル」の「Interface CardBus ID Utility」を開きます。
2. 現在挿入されている弊社CardBus製品の情報が表示されます。左から「バス番号」、「デバイス番号」、「ファンクション番号」、「型式」、「ID番号」を示します。ID番号の設定を行いたい製品をダブルクリックして下さい。
3. ID番号を設定するダイアログが開きますので、コンボボックス内で設定したいID番号を選択し、「OK」ボタンをクリックします。
4. これでID番号の設定は完了です。

※ 設定できるID番号は0h～Fhまでです。
※ 変更したID番号をドライバに認識させるには、一度CardBus製品を抜き差しするか、システムを再起動して下さい。
※ 設定したID番号が分かるように番号を記したシールをCardBus製品に貼ることをお勧めします。

第7章　重要な情報

保証の内容と制限

弊社は本ドキュメントに含まれるソースプログラムの実行が中断しないこと、またはその実行に誤りが無いことを保証していません。
本製品の品質や使用に起因する、性能に起因するいかなるリスクも使用者が負うものとします。

弊社はドキュメント内の情報の正確さに万全を期しています。万一、誤記または誤植などがあった場合、弊社は予告無く改訂する場合があります。ドキュメントまたはドキュメント内の情報に起因するいかなる損害に対しても弊社は責任を負いません。
ドキュメント内の図や表は説明のためであり、ユーザ個別の応用事例により変化する場合があります。

著作権、知的所有権

弊社は本製品に含まれるおよび本製品に対する権利や知的所有権を保持しています。
本製品はコンピュータ　ソフトウェア、映像/音声(例えば図、文章、写真など)を含んでいます。

医療機器/器具への適用における注意

弊社の製品は人命に関わるような状況下で使用される機器に用いられる事を目的として設計、製造された物では有りません。
弊社の製品は人体の検査などに使用するに適する信頼性を確保する事を意図された部品や検査機器と共に設計された物では有りません。
医療機器、治療器具などの本製品の適用により、製品の故障、ユーザ、設計者の過失などにより、損傷/損害を引き起こす場合が有ります。

複製の禁止

弊社の許可なく、本ドキュメントの全て、または一部に関わらず、複製、改変などを行うことはできません。

責任の制限

弊社は、弊社または再販売者の予見の有無にかかわらず発生したいかなる特別損害、偶発的損害、間接的な損害、重大な損害について、責任を負いません。

本製品(ハードウェア，ソフトウェア)のシステム組み込み、使用、ならびに本製品から得られる結果に関する一切のリスクについては、本製品の使用者に帰属するものとします。

本製品に含まれるバグ、あるいは本製品の供給（納期遅延）、性能もしくは使用に起因する付帯的損害もしくは間接的損害に対して、弊社に全面的に責がある場合でも、弊社はその製品に対する改良（正常に動作する）、代品交換までとし、金銭面での賠償の責任は負わないものとしますので、予めご了承ください。

本製品（ソフトウェアを含む）は、日本国内仕様です。本製品を日本国外で使用された場合、弊社は一切責任を負いかねます。また、弊社は本製品に関し、海外での保守サービスおよび技術サポート等は行っておりません。

商標/登録商標

本書に掲載されている会社名、製品名は、それぞれ各社の商標または登録商標です。